N° d'ordre : 486 *N° attribué par la bibliothèque : 07ENSL0486*

THÈSE

pour obtenir le grade de

Docteur de l'Université de Lyon - École Normale Supérieure

spécialité : Informatique

Laboratoire de l'Informatique du Parallélisme

École doctorale de Mathématiques et Informatique fondamentale

présentée et soutenue publiquement le 31 Octobre 2008 par

Monsieur Raphaël BOLZE

Analyse et déploiement de solutions algorithmiques et logicielles pour des applications bioinformatiques à grande échelle sur la grille

Directeur de thèse : Monsieur Frédéric DESPREZ

Après avis de : Monsieur Vincent BRETON
Monsieur Pierre MANNEBACK

Devant la commission d'examen formée de :

Monsieur Vincent	BRETON	Membre/Rapporteur
Monsieur Frédéric	DESPREZ	Membre
Monsieur Pierre	MANNEBACK	Membre/Rapporteur
Monsieur Thierry	PRIOL	Membre/Président du jury
Monsieur Jacques	DELPLANCQ	Membre
Monsieur Thierry	TOURSEL	Membre

Résumé

Cette thèse présente un ensemble d'objectifs dont le fil conducteur est le programme Décrypthon (projet tripartite entre l'AFM, le CNRS et IBM) où les applications et les besoins ont évolué au fur et à mesure de l'avancée de nos travaux. Dans un premier temps nous montrerons le rôle d'architecte que nous avons endossé pour la conception de la grille Décrypthon. Les ressources de cette grille sont supportées par les cinq universités partenaires (Bordeaux I, Lille I, ENS-Lyon, Pierre et Marie Curie Paris VI et Orsay), ainsi que le réseau RENATER (Réseau National de Télécommunications pour l'Enseignement et la Recherche), sur lequel est connecté l'ensemble des machines. Le Centre de ressources informatiques de Haute Normandie (CRIHAN) participe également au programme, il héberge les données volumineuses des projets scientifiques. Nous présenterons ensuite les expériences que nous avons effectuées sur l'intergiciel DIET afin de tester ses propriétés de façon à explorer sa stabilité dans un environnement à grande échelle comme Grid'5000. Nous nous sommes intéressés, en outre, au projet "Help Cure Muscular Dystrophy", un des projets sélectionnés par le programme Décrypthon. Nous avons conduit des expériences dans le but de préparer la première phase de calcul sur la grille de volontaires "World Community Grid". Nous dévoilerons l'ensemble des étapes qui ont précédées et suivies la première phase calculatoire qui a demandé quelques 80 siècles de temps processeur. Pour terminer, nous avons développé une fonctionnalité à l'intergiciel DIET, le rendant capable de gérer l'exécution de tâches ayant des dépendances. Nous nous sommes intéressés à développer des algorithmes prenant en compte plusieurs applications qui demandent l'accès aux mêmes ressources de manière concurrente. Nous avons validé cette fonctionnalité avec des applications issues des projets du programme Décrypthon. Ces travaux ont nécessité un développement logiciel important, d'une part sur les applications du Décrypthon elles-mêmes et sur leur portage afin de rendre transparente leur utilisation sur la grille Décrypthon, mais aussi au niveau de l'intergiciel DIET et son écosystème : DIET_Webboard, VizDIET, GoDIET, LogService, MA_DAG, *etc.* Les résultats présentés ont été obtenus sur trois grilles mises à notre disposition : la grille universitaire du Décrypthon, la grille d'internautes (World Community Grid) et la grille expérimentale Grid'5000.

Abstract

This thesis was conducted by the needs of the Decrypthon project (collaborative project between AFM, CNRS and IBM). First we show the role of architect played in order to select and define the Decrypthon grid infrastructure. The resources of this grid are hosted by five Universities (Bordeaux I, Lille I, ENS-Lyon, Pierre et Marie Curie Paris VI et Orsay). The network connexion is provided by RENATER (Réseau National de Télécommunications pour l'Enseignement et la Recherche). The CRIHAN (Centre de ressources Informatiques de Hautes Normandie) is also involved into this parternship and provides data warehouse for scientists. In a second hand we present several experiments carried on Grid'5000 in order to validate the grid middleware DIET and its tools on a large scale platform such as Grid'5000. On this research platform, we also studied the application of the project "Help Cure Muscular Dystrophy", one of the project selected by the Decrypthon. This study prepared the launch of a 6 months computing phase on the volunteers grid : World Community Grid support by IBM US. The document presents all steps before and after the computing phase which require more than 80 centuries of CPU time on the volunteers device. Finally, we have designed several heuristics to tackle the problem of online multi-workflow scheduling in a shared grid environment. We have implemented those heuristics into DIET middleware and we have validated their behavior with case study applications from Decrypthon. This work required many software developments in the aim to grid enabled bioinformatic applications and transparenlty give access to the Decrypthon grid, but also into DIET middleware and tools around : DIET_Webboard, VizDIET, GoDIET, LogService, MA_DAG, *etc.* The results exposed in this thesis were obtained with tree different grids : the Decrypthon grid, the volunteer grid (World Community Grid) and the research grid (Grid'5000).

Remerciements

Je commencerais par remercier Frédéric Desprez, mon directeur de thèse pour m'avoir donné la chance de faire une thèse sous sa direction. Il a cru en mes capacités, et il m'a donné les opportunités pour développer et prouver mes compétences. C'est grâce à lui que j'ai eu la chance de m'investir dans l'aventure du programme Décrypthon. Il m'en a confié la responsabilité officieuse et je lui en suis reconnaissant. Malgré nos entretiens parfois espacés en raison de ses nombreuses responsabilités, il a toujours gardé un rôle de conseil et de guide. J'ai souvent eu du mal à croire en mon travail, ayant toujours le sentiment que je pouvais en faire plus. Le recul me donne à présent la mesure de ce que j'ai pu accomplir sous ta direction : Merci Fred.

Par ailleurs, l'écriture de cette thèse s'inscrit au sein d'une équipe que nous aimons appeler : la « DIET team ». Je dois dire que c'est en partie grâce à ce contexte et cet environnement stimulant que j'ai fourni ce travail. La bonne humeur et la circulation des idées favorisent la créativité de chacun. Merci aux permanents Eddy, Fred, Yves, aux anciens de l'équipe, Alan, Holly, Bruno, Abdelkader, Antoine, Éric, Matthieu, comme aux autres David, Nicolas, Benjamin, Ben, Gislain, Gaël, Cédric, JS. Plus généralement, toutes les personnes de l'équipe GRAAL Loris, Yves, Fredo, Anne, Jean-Yves, Manu, Mathieu, Jeff, Véronika et toutes les personnes que j'ai pu connaître (Abdou, LN, Arnaud, *etc*) qui ont fait que je m'y sente bien.

En outre, cette thèse s'est déroulée dans le cadre du programme Décrypthon. Si l'envie vous prend de lire la suite du manuscrit, vous découvrirez l'historique et le contexte de ce projet. Je remercie tous les acteurs du projet dans lequel j'ai réellement pris du plaisir à découvrir et à échanger des savoirs dans des domaines que je ne connaissais pas avant. Je pense notamment à Christophe lorsqu'il m'a montré et expliqué ses expériences sur les blattes jusqu'à l'exploitation statistique de ces données. Je pense aussi à Anne, Luc et Olivier avec qui j'ai eu du plaisir à travailler et à appréhender le langage particulier d'un bio-informaticien. Il y a eu aussi les nombreux intervenants d'IBM, notamment le duo de choc Fabien et Emmanuel, mais aussi Anne-Marie, Karine, Marie-Hélène, Pierre-Jean et Hervé avec qui j'ai appris à travailler dans un contexte différent de celui de la recherche. Merci à tous.

Un grand merci aux relecteurs de l'ombre : Jean-Thomas, Amélie et mes beaux-parents. Ils ont contribué à corriger les fautes et parfois alléger le style de mes phrases.

Par ailleurs, je voudrais consacrer un paragraphe à démentir quelques rumeurs appelées aussi « Raphaël Bashing ». Il est clair que je ne cacherai pas mon caractère qui ne se satisfait pas d'une défaite. Je joue pour gagner,

et même si j'accepte les quelques défaites concédées, je ne m'en contenterai jamais. A ce propos, je voudrais rétablir les quelques contre-vérités qui se sont glissées dans les remerciements d'autres manuscrits : non, Antoine je ne te laisserai pas parler de « branlées » et surtout si elles avaient eu lieu, je n'en serais pas vexé. Non Loris, mes défaites ne sont pas tragiques et encore moins systématiques. Non Fredo, tu n'as pas besoin de me ménager, tu pouvais en tant que chef d'équipe battre un thèsard. Aussi, Yves, tu resteras un adversaire à la hauteur de mon acharnement dans la recherche d'une victoire et je ne désespère pas un jour de te battre (régulièrement). Enfin, vous aurez compris que comme dans toutes les rumeurs, il y a une part de vérité, et sans aller jusqu'à remercier les instigateurs, je leur en témoigne mon amitié « fair-play ».

Enfin, mon dernier remerciement revient à ma famille et particulièrement ma femme, Amélie, qui m'a épaulé tout au long de ma thèse.À la fois, parce qu'elle m'a laissé travailler souvent plus que de raison, mais aussi pour son soutien inconditionnel et sa présence à mes côtés. Merci d'être toi tout simplement.

Table des matières

Introduction

Le traitement parallèle des calculs et des données a gagné toutes les sciences. Hier, réservé aux applications gourmandes en temps de calcul issues notamment de la simulation en mécanique ou en météorologie, il a rejoint maintenant d'autres applications aux besoins différents, comme par exemple l'imagerie médicale, la modélisation « in silico » d'interactions médicamenteuses ou même la recherche de documents sur Internet. Ces applications ont nécessité de nouvelles approches, pas forcement liées à une utilisation efficace des caches de la mémoire, des processeurs et autres spécificités des ordinateurs.

Du côté des architectures matérielles, nous sommes passés des gros calculateurs propriétaires comme les machines Cray, NEC, IBM, *etc*, à des grappes montées à partir d'ordinateurs personnels (PC) interconnectés par des réseaux rapides. Nous avons vu ensuite ces grappes s'agréger en grappes de grappes puis, depuis le début des années 2000, une mise en commun globale des moyens de calcul et de stockage, que ce soit avec des machines énormes de centres de calcul ou des PCs de bureau. Après des débuts chez les universitaires, cet agrégat de ressources appelé « grilles informatiques » trouvent maintenant une place grandissante dans les entreprises soucieuses de rationaliser et de rentabiliser leurs investissements informatiques. Par ailleurs, la démocratisation des ordinateurs personnels et des connexions Internet chez les particuliers a fait pénétrer les technologies de grilles jusque dans les foyers. Ainsi, le site internet [31] référence plusieurs dizaines de projets de calcul supportés par des internautes volontaires.

Les capacités de calcul de ces plates-formes diffèrent énormément, à la fois en terme de nombre d'opérations par seconde (nous dépassons le Teraflops) mais aussi en fonction de leur disponibilité ou de leur capacité à échanger des données. Les scientifiques d'aujourd'hui, qu'ils soient issus de laboratoires de recherche dans toutes les sciences ou de grandes entreprises privées, ont accès à des moyens informatiques autrefois réservés à des centres nationaux.

Le problème commun à toutes ces architectures n'est donc pas le matériel mais plutôt le logiciel permettant d'exploiter cette puissance (du système à l'algorithmique). En effet, il y a peu de chance que ces machines soient totalement homogènes. Au mieux, nous agrégeons des machines de même type par un réseau rapide et à l'opposé, nous prenons la puissance de calcul disponible sur la toile et, des réseaux aux processeurs en passant par les systèmes d'exploitation, ainsi l'ensemble de l'architecture devient hétérogène.

Il y a deux challenges principaux pour le développement des plates-formes

de grilles : le développement d'**environnements** standards rendant transparente l'utilisation de la grille et des résultats sur l'**algorithmique** des applications utilisant de telles plates-formes. Dans « environnement », nous pouvons placer bien sûr le système d'exploitation mais aussi les langages, les bibliothèques et l'intergiciel. Actuellement, la plupart des environnements existants reprennent les approches choisies pour les machines parallèles « classiques » en tentant de les adapter aux concepts de la grille. Les résultats sont bien entendu inégaux.

Un domaine d'application important des grilles concerne la bioinformatique : le domaine des sciences où la biologie, l'informatique et les mathématiques se rejoignent pour former une seule et unique discipline. Au centre de nombreuses recherches dans le monde, que ce soit dans la recherche académique que dans l'industrie de la santé, ce domaine, à l'impact sociétal important, est appelé à se développer de manière très importante dans un futur très proche. Les caractéristiques de ces applications diffèrent de celles du calcul numérique « classique » qui s'exécute actuellement en masse sur les grilles de production du monde entier. Elles posent donc de nouveaux problèmes de recherche informatiques importants auxquels nous nous sommes intéressés dans cette thèse.

En 2001, le projet Décrypthon, a été en France un des pionniers dans ce domaine en mettant à contribution les internautes pour cartographier le protéome. Grâce à ce succès, en 2005, l'AFM associée au CNRS et à IBM, a renouvelé son action avec le *programme Décrypthon*. Il vise, cette fois, à aider plusieurs projets scientifiques des sciences du vivant à accéder aux technologies de grille en leur apportant à la fois des ressources de calculs réparties dans des centres de calculs universitaires mais également une expertise dans la « gridification » des applications.

Un sujet de recherche commun pour le portage d'applications sur les grilles reste l'ordonnancement de tâches sur des ressources hétérogènes. Que ce soit sur des plates-formes dédiées ou sur des grilles d'internautes, il faut être capable d'orchestrer au mieux les tâches (qu'elles soient dépendantes ou pas) sur les ressources et également choisir le grain de calcul et de communication qui permettra d'obtenir les meilleures performances. Ce sujet a été évidemment étudié par de nombreux chercheurs de part le monde avec des modèles plus ou moins réalistes.

Nous nous sommes attachés à étudier un problème classique des applications parallèles, l'ordonnancement de graphes de tâches (aussi appelés par abus de langage *workflows*) mais cette fois en supposant que plusieurs utilisateurs peuvent envoyer de tels graphes sur la plate-forme de grille cible. Il faut alors non seulement optimiser le temps de calcul global de chaque application mais aussi l'utilisation de la plate-forme et l'équité entre les utilisateurs.

Du côté des plates-formes d'internautes, nous nous sommes intéressés non seulement au portage efficace d'une application de taille conséquente sur la grille du World Community Grid mais également à une comparaison avec les grilles dédiées.

Ces travaux ont nécessité un développement logiciel important, d'une part sur les applications elles-mêmes et sur leur portage afin de rendre transparente leur utilisation sur la grille Décrypthon, mais aussi au niveau de l'intergiciel DIET et son écosystème : DIET_Webboard, VizDIET, GoDIET, LogService, MA_{DAG}, *etc.* Les résultats présentés ont été obtenus sur trois grilles mises à notre disposition : la grille universitaire du Décrypthon, la grille d'internautes (World Community Grid) et la grille expérimentale Grid'5000.

Ce manuscrit présente un ensemble d'objectifs et de résultats dont le fil conducteur a été le programme Décrypthon dans lequel les applications et les besoins ont évolué au fur et à mesure de l'avancée de nos travaux. Dans un premier temps nous montrerons le rôle d'architecte et d'accompagnateur des projets scientifiques que nous avons endocé au niveau de la grille universitaire Décrypthon [7,3].

Nous présenterons ensuite les expériences que nous avons effectuées sur l'intergiciel DIET afin de tester ses propriétés de façon à explorer sa stabilité dans un environnement à grande échelle [1,2,4,5]. Nous nous sommes intéressés, en outre, au projet *Help Cure Muscular Dystrophy*, un des projets sélectionnés par le programme Décrypthon. Nous avons conduit des expériences dans le but de préparer le lancement du projet HCMD sur une grille de volontaires [6]. Nous dévoilerons l'ensemble des étapes qui ont précédées et suivies la première phase calculatoire qui a demandé quelques 80 siècles de temps processeur sur les ressources des membres du World Community Grid.

Enfin, nous avons ajouté une fonctionnalité à l'intergiciel DIET, le rendant capable de gérer l'exécution de tâches ayant des dépendances. Nous nous sommes intéressés à développer des algorithmes prenant en compte plusieurs applications qui demandent l'accès aux mêmes ressources de manières concurrentes. Nous avons validé cette fonctionnalité avec des applications issues des projets du programme Décrypthon. L'ensemble des développements autour de la gestion des graphes de tâches est disponible dans la version courante de DIET (version 2.3).

Publications

Nos travaux ont été publiés dans 3 journaux internationaux, 3 conférences internationales et 1 conférence nationale. Elles valident les résultats obtenus au cours de cette thèse et marque le travail collaboratif et pluridisciplinaire.

Journaux Internationaux (avec comité de lecture)

[1] A. Amar, R. Bolze, Y. Caniou, E. Caron, B. Depardon, J.S. Gay, G. Le Mahec, and D. Loureiro. Tunable scheduling in a GridRPC framework. *Concurrency & Computation : Practice & Experience*, 20(9) :1051-1069, 2008.

[2] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E. Talbi, and T. Irena. Grid'5000 : a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4) : 481-494, November 2006.

[3] N. Garnier, A. Friedrich, R. Bolze, E. Bettler, L. Moulinier, C. Geourjon, J. D. Thompson, G. Deleage, and O. Poch. MAGOS : multiple alignment and modelling server. *Bioinformatics*, 22(17) : 2164-2165, 2006.

Conférences Internationales (avec comité de lecture)

[4] A. Amar, R. Bolze, A. Bouteiller, P. K. Chouhan, A. Chis, Y. Caniou, E. Caron, H. Dail, B. Depardon, F. Desprez, J.-S. Gay, G. Le Mahec, and A. Su. Diet : New developments and recent results. *CoreGRID Workshop on Grid Middleware* ,in conjunction with EuroPar2006, number 4375 in LNCS, pages 150-170, Dresden, Germany, August 28-29 2006. Springer.

[5] V. Bertis, R. Bolze, F. Desprez, and K. Reed. Large scale execution of a bioinformatic application on a volunteer grid. *Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC)*, in conjuction with IPDPS, April 2008.

[6] R. Bolze, E. Caron, F. Desprez, G. Hoesch, and C. Pontvieux. A monitoring and visualization tool and its application for a network enabled server platform. *Computational Science and Its Applications - ICCSA 2006*, volume 3984 of LNCS, pages 202-213, Glasgow, UK., May 8-11 2006.

Conférences nationales

[7] P. d'Anfray, R. Bolze, and F. Desprez. Le programme Décrypthon. *Journée Réseaux JRES*, Strasbourg, 2007.

Chapitre 1

Qu'est-ce qu'une grille informatique

Sommaire

1.1 Introduction

En informatique, certains algorithmes peuvent être divisés en calculs élémentaires afin d'être répartis entre plusieurs processeurs. L'intérêt de cette séparation est de pouvoir faire opérer des processeurs simultanément pour pouvoir traiter plus rapidement le calcul que lors d'une version séquentielle de l'algorithme. C'est l'idée générale du calcul parallèle.

Le parallélisme est aussi vieux que l'informatique et l'écriture de programmes. Rapidement, la parallélisation des architectures matérielles s'est effectuée en agrégeant plusieurs processeurs entre eux *Symmetric Multi Processeur* (SMP). Dans un même temps, avec la vente en masse des ordinateurs, la baisse des prix et le développement des protocoles de communication intermachines, l'idée d'agréger des machines entre elles pour former des grappes de machines est apparue. Dans cet effort, en 1993, le projet Beowulf [102] bouleversa le monde des machines parallèles, qui était jusqu'alors réservées aux centres de calcul capables d'acquérir des supercalculateurs basés sur des technologies propriétaires aux coûts élevés.

Les besoins en calculs scientifiques ont évolué avec les technologies disponibles. Et parallèlement aux évolutions matérielles, les outils permettant la programmation des machines ont permis d'exploiter le parallélisme de l'application au moment même de son implantation. Le paysage est maintenant rempli d'une multitude d'outils en passant par les systèmes d'exploitation (XtremOS, Vigne, GridOS, *etc*), les langages de programmation (HPF, C/C++, Java, *etc*) et leurs compilateurs, les bibliothèques de communications (MPI, PVM), les bibliothèques de calcul (ScaLAPACK, MUMPS, PBLAS, *etc*) et les interfaces de programmation (openMP, POSIX, *etc*) permettant de construire une application tirant partie du parallélisme.

1.2 Définitions d'une grille informatique

1.2.1 Un ensemble de définitions

Ian Foster et Carl Kesselman ont posé les premières pierres de l'édifice de la recherche sur les grilles informatiques qui à cette époque s'appelaient *metacomputing*. Il y a 10 ans, dans leur livre « *The Grid : Blueprint for a New Computing Infrastructure* », ils ont fait l'analogie du partage des ressources informatiques avec l'accès à l'électricité. Ainsi, ils établissent la vision d'un utilisateur qui se branche et obtient la ressource dont il a besoin sans pour autant connaître tout ce qui se cache derrière. Plus tard, ils ont proposé une définition de la grille informatique basée sur 3 critères :

- une grille n'a pas de gestion centralisée de l'ensemble de ses ressources informatiques ;
- une grille repose sur des protocoles standardisés ;
- une grille offre une qualité de services.

Cette définition n'est pas précise : Quels sont les protocoles ? Quels sont les services ? C'est certainement ce flou autour de la définition qui a fait son succès. Chacun peut ainsi spécifier sa définition en précisant comment s'effectue la gestion non centralisée des ressources, quels sont les standards et protocoles utilisés, et quelles qualités de services sont offertes par la grille.

Le terme de « grille informatique » est employé pour décrire des concepts qui cachent souvent une réalité et des significations différentes. L'encyclopédie en ligne wikipedia [63] donne la définition suivante :

> *Une grille informatique est une infrastructure virtuelle constituée d'un ensemble de ressources potentiellement partagées, distribuées, hétérogènes, dé-localisées et autonomes.*

Le terme *potentiellement* donne tout son sens à la définition : Il existe plusieurs grilles informatiques. Elles n'ont pas nécessairement toutes les propriétés énoncées dans la définition mais des dénominateurs communs existent.

1.2.2 Les différentes classifications

Malgré la définition que nous avons donnée d'une grille informatique, les latitudes de mise en œuvre sont grandes. Une taxinomie souvent employée [52, 58] distingue deux classes de grilles :

- les grilles de calcul : mise en commun de ressources dans le but de faire des calculs ;
- les grilles de données : mise en commun de ressources pour stocker de l'information.

Ces deux catégories mettent en avant une caractéristique des ressources informatiques qui ont été agrégées. Cependant, il est clair que l'on ne peut pas envisager les calculs sans capacité de stockage. De même, une grille de données demande des ressources de calcul pour traiter les informations qu'elle contient. Les grilles informatiques ne se résument pas donc à ces deux classes.

Ainsi, la présentation d'une grille informatique se fait par rapport aux ressources, aux logiciels qui sont utilisés, aux protocoles déployés et aux différents services disponibles. Tous ces points font appel à la définition d'un point de vue souvent très technique qui va mettre en avant un caractère particulier avec une technologie particulière. Toutes les tentatives de classification [58, 52] adoptent une vision différente suivant l'orientation suivie. Nous

opterons ici pour une énumération (non exhaustive) des différents points de vue que nous pouvons rencontrer :

- la finalité de la grille : il peut s'agir d'une grille de production (la grille est utilisée dans le but de fournir un résultat), d'une grille de test/recherche (utilisée dans le but d'évaluer les fonctionnalités de programmes, d'intergiciels, de protocoles réseaux) ;
- l'objectif d'utilisation de la grille : il peut s'agir d'une grille de calcul (le but premier est d'effectuer des calculs), d'une grille de stockage (mise en commun de machines dédiées au stockage d'énormes quantités de données) ;
- les applications scientifiques qui utilisent la grille. Elles sont nombreuses. L'omniprésence de l'informatique dans les sciences modernes entraîne chaque jour de nouveaux domaines à investir les calculs distribués lorsque les ressources d'une seule machine ne suffisent plus. Nous sommes à l'ère de l'*e-science*. Citons les domaines de manière générale, les mathématiques, les sciences de la vie et de la terre, la chimie, la physique, mais nous pouvons rentrer dans le détail en parlant de cryptographie, de génomique, de climatologie, de cristallographie, de la physique des particules, *etc.*
- les technologies employées. C'est dans ce domaine que l'on retrouve le plus grand nombre de points de vue différents. Il existe presque autant de technologies différentes qu'il a pu être construit de types d'ordinateurs. Elles évoluent continuellement et elles suivent les tendances et les innovations : citons les « grilles pair-à-pair », les « grilles à service web », les « grilles à composants », les « grilles client-serveur », *etc.*
- les ressources mises en œuvre : des machines parallèles, des grappes de machines, des ordinateurs personnels. Les termes employés sont alors « grille de supercalculateurs », « grille de grappes », « grille d'ordinateurs personnels », *etc.*
- les mécanismes de sélection des ressources : modèle *pull* (tirer) les ressources viennent chercher les travaux à effectuer, modèle *push* (pousser) les travaux sont envoyés sur les ressources. L'environnement est appelé :« grille à vol de cycles », « grille partagée », « grille dédiée », *etc.*

Tous ces points de vue se retrouvent donc dans l'ensemble des grilles qui composent le paysage du calcul scientifique.

À l'image du modèle OSI [72] pour les protocoles réseaux, un modèle en quatre couches logiques est proposé [53] pour les grilles informatiques. La figure 1.1 illustre cette décomposition. Sous la couche *application* qui utilisera les ressources, trois couches se distinguent : la couche *intergiciel* et la couche *service* que nous détaillerons par la suite. La couche *infrastructure*, quant à

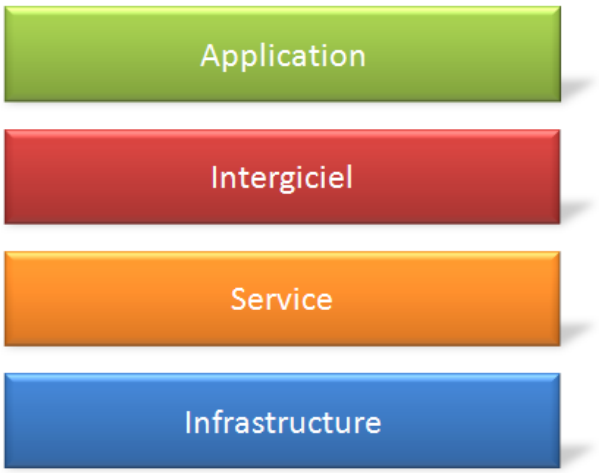


Figure 1.1 – Modèle en couche des grilles.

elle, représente l'ensemble des ressources mises en œuvre pour la construction d'une grille.

1.2.3 Ressources d'une grille informatique

Quatre types de ressources composent les éléments principaux de l'infrastructure d'une grille informatique :

Ressources de communication : elles regroupent tous les éléments qui permettent la communication entre les différents composants de la grille. Ce sont les ressources réseaux comme les concentrateurs (*hub*), les commutateurs (*switch*), les routeurs, *etc.* Elles englobent aussi les liens d'interconnection. Ils sont caractérisés par les protocoles utilisés (ATM, FDDI, MPLS, Ethernet, Myrinet, *etc*) qui détermineront leur capacité de transfert, leur débit, leur latence, *etc.*

Ressources de calcul : ce sont tous les éléments qui permettent l'exécution d'une application ou d'un code. Le représentant de ces ressources est bien entendu le processeur d'une machine, sachant qu'il peut y avoir plusieurs processeurs par machine et même plusieurs cœurs par processeur, sans oublier les processeurs dédiés comme le processeur graphique. Ces ressources sont caractérisées par leur vitesse de traitement, c'est à dire le nombre d'opérations en virgule flottante (FLOPS/s), leur consommation d'énergie, leur précision, *etc* ;

Ressources de stockage : elles distinguent tous les lieux de stockage de données qui sont utilisés. Il en existe plusieurs types avec des caractéristiques différentes et des utilisations différentes. Par exemple, la mémoire vive disponible au niveau des processeurs, ou, plus finement, les différents niveaux de cache (L1, L2, L3...) d'un processeur, ou encore l'espace de stockage sur le disque dur propre à chaque ordinateur, ou encore les systèmes de fichiers partagés d'une grappe (NFS, GPFS, HDFS, *etc*). Suivant la ressource de stockage considérée, elle peut être partagée, répartie, ou encore locale. Elle se caractérise par sa capacité, c'est-à-dire sa taille (en octet), son débit en lecture et écriture, sa latence d'accès, sa consommation énergétique ;

Ressources humaines : ces dernières sont souvent oubliées, voire ignorées. Elles représentent cependant le maillon essentiel dans la vie et l'exploitation d'une grille. En effet, une grille informatique repose sur la coopération humaine de différentes personnes en charge des ressources et sur la communauté utilisatrice de la grille.

1.2.4 Un intergiciel de grille et des services

Une définition générale est donnée par wikipedia [64] :

> *Un intergiciel est un logiciel servant d'intermédiaire entre plusieurs applications ou services, généralement complexes et distribuées sur un réseau informatique.*

Il s'agit de définir les éléments de base d'un intergiciel dans le contexte des grilles informatiques. D'après la figure 1.1, la place occupée par l'intergiciel est fondamentale. Il joue le rôle d'intermédiaire entre les applications qui utiliseront les services et les ressources. La frontière des rôles qui lui sont attribués s'enchevêtre souvent avec les services qu'il rend. Quels sont les applications ou les services mis en relation par un intergiciel de grille ?
Ces services peuvent être regroupés en 4 thèmes :

- la sécurité ;
- la gestion des données ;
- la gestion de l'exécution ;
- l'information ;

À la fin des années 90, lorsque Ian Foster et son équipe ont proposé une définition de la grille, ils ont aussi proposé une implantation sous forme d'un ensemble de composants pour construire une grille informatique. Nous exposons ici l'implantation du *Globus Toolkit*, qui représente une illustration de la définition d'un intergiciel de grille.

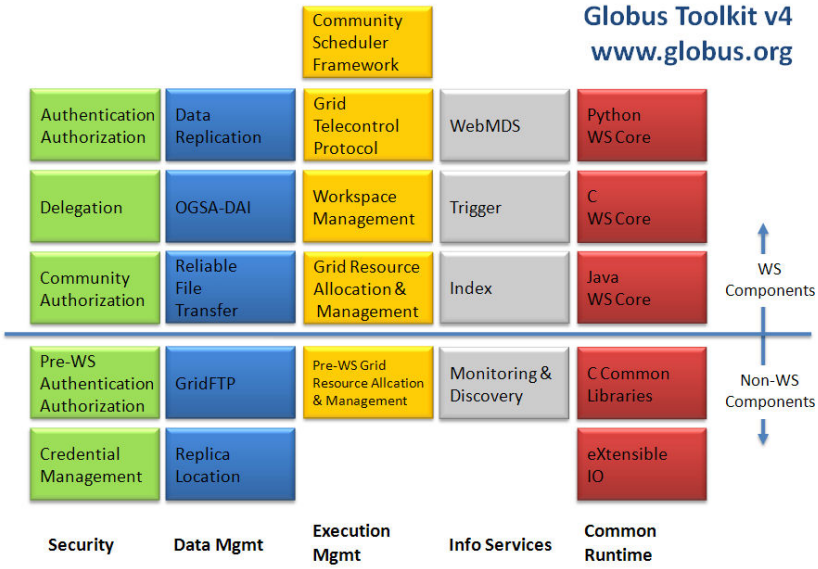


Figure 1.2 – Les briques de base définies par le *Globus Toolkit*.

Globus Toolkit

Le *Globus Toolkit* [3] est un ensemble de programmes et de bibliothèques qui s'efforce de résoudre les difficultés de construction d'un environnement distribué de services et d'applications. Le *Globus Toolkit* est organisé de façon modulaire. Il comprend un ensemble de composants qui implantent les services de base pour la sécurité, la communication, l'allocation de ressources, *etc.* L'utilisation et la combinaison de ces différents composants permet de configurer l'intergiciel de grille afin qu'il soit adapté aux besoins particuliers de ceux qui souhaitent construire leur propre grille.

Le *Globus Toolkit* propose un ensemble de briques logicielles. Elles délimitent l'intergiciel de la grille. Une des grandes force de l'Alliance Globus est d'avoir défini l'ensemble des composants d'un intergiciel de grille et d'avoir proposé une implantation de chacun d'eux de manière modulaire, sans pour autant les rendre dépendants les uns des autres.

- *Globus Resource Allocation Manager* (GRAM) : il a pour rôle de convertir des requêtes vers la grille en demandes compréhensibles par les différentes ressources qui la composent ;
- *Grid Security Infrastructure* (GSI) : il établit la sécurité et l'authentification des utilisateurs ;
- *Monitoring and Discovery Service* (MDS) : il collecte et stocke des informations sur les différentes ressources (capacité de stockage, puissance

de calcul, bande passante, ...).
- *Grid Resource Information Service* (GRIS) : il maintient l'état des ressources, leur configuration et leur capacité.
- *Grid Index Information Service* (GIIS) : il coordonne les différents services GRIS et permet notamment de faire des recherches sur les caractéristiques des ressources.
- *GridFTP* : il fournit un service de transfert robuste, fiable et rapide.
- *Replica Catalog* (RC) : il maintient une liste des différents emplacements dans lesquels sont stockés les fichiers ainsi que leurs copies.
- *Replica Management System* (RMS) : il permet aux applications de créer des copies de certaines données et de les gérer.

Chacun peut tout à fait n'utiliser qu'une partie des briques implantées dans le *Globus Toolkit* afin de construire son propre intergiciel. La figure 1.2 illustre le découpage suivant les 4 domaines : sécurité, stockage, exécution et information. Par exemple, gLite en réutilisant certains éléments du *Globus Toolkit* et en développant ses propres composants, est devenu un intergiciel à part entière.

1.3 Quelques exemples de grilles

Nous présentons ici quelques exemples de grilles informatiques, en détaillant les ressources employées et l'intergiciel qui les exploite. Nous avons décidé de présenter uniquement 3 grilles différentes. Elles représentent un tout petit aperçu représentatif des grilles actuellement mises en place dans le monde académique.

1.3.1 EGEE

L'EGEE (***E**nabling **G**rids for **E**-scienc**E***) [76] est un projet financé par la Commission européenne depuis mars 2004. Il fait suite au projet DATAGrid [89]. Actuellement dans sa phase III depuis le 1er mai 2008, son but est de contribuer à l'avancement des technologies et de mettre en place une grille de calcul et de stockage à l'échelle mondiale. Cette grille met en synergie des chercheurs et ingénieurs issus de domaines et d'horizons différents. Elle est développée suivant trois objectifs principaux :
- construire une grille de calcul et de stockage cohérente, robuste et sécurisée ;
- améliorer de manière continue la qualité du logiciel pour fournir un service fiable aux utilisateurs ;

– attirer de nouveaux utilisateurs provenant du domaine scientifique ou de l'industrie, en leur garantissant des performances de haut niveau ainsi que le support nécessaire.

Au travers des trois composantes développées par ce projet d'envergure européenne, nous notons le souci d'inscrire cette grille informatique dans la notion de qualité de services pour les utilisateurs. Il s'agit d'un des trois critères donnés par Ian Foster et Carl Kesselman pour définir une grille.

Le travail est divisé en trois activités, elles-mêmes découpées en domaines :

– *Networking Activities* (NA) : cinq domaines d'activité sont dénombrés ; l'activité de coordination globale au projet (NA1), de diffusion de l'information pour atteindre de nouvelles communautés (NA2), de formation des utilisateurs et de production du matériel pour cette formation (NA3), d'identification et du support des applications (NA4) et enfin la gestion de la coopération à l'international (NA5).
– *Service Activities* (SA) : elle-même répartie en trois domaines, l'activité de support de la grille européenne (SA1), la mise à disposition des ressources du réseau (SA2) et l'activité d'intégration, test et certification (SA3).
– *Joint Research Activities* (JRA) : elle-même décomposée en deux domaines. Les activités couvrent l'intégration et le développement des logiciels (JRA1), l'assurance qualité (JRA2).

GLite : l'intergiciel de la grille EGEE

GLite se veut l'intergiciel de nouvelle génération d'EGEE. Il est né de la contribution de plus de 80 personnes issues d'au moins 12 instituts différents. Il se base sur les connaissances et projets logiciels et intergiciels qui ont été développés par le passé.

Plus particulièrement nous retrouvons dans gLite les composants suivants :

– *Computing Element* (CE) : il représente un ensemble de ressources de calcul, typiquement une grappe de serveurs. Il peut aussi représenter une seule machine.
– *Worker Nodes* (WN) : il représente les nœuds (ou machines) d'une grappe de serveurs.
– *Storage Element* (SE) : il permet un accès centralisé à un ensemble de ressources de stockage. Tout site possède en général un SE.
– *Grid Security Infrastructure* (GSI) : il assure la sécurité et l'authentification des utilisateurs, ce composant est emprunté au *Globus Toolkit*.
– *Virtual Organisation Membership Service* (VOMS) : il s'agit d'un service destiné à la gestion des informations des utilisateurs et leur appar-

tenance à une organisation virtuelle. Il gère les certificats en provenance du *GSI* ;
- *Information Service* (IS) : le service de monitoring de la grille basée sur les ressources, il permet de répertorier et de contrôler chacune des ressources, comme les *Storage Elements* ou les *Computing Elements*.
- *Logging and Bookkeeping* (LB) : composant qui permet de stocker l'activité des différents modules de gLite. Le *Workload Management System* (WMS), l'*interface utilisateur* (UI), le *Local Resource Management System* (LRMS) ou *Computing Element* (CE) transmettent chacun des informations à ce composant ;
- *Job Provenance Service* (JP) : il est en charge du stockage à long terme des informations des travaux des utilisateurs à partir des informations recueillies avec le *Logging and Bookkeeping* (LB) ;
- *User Interface* (UI) : le point d'accès à la grille pour un utilisateur ;
- *Workload Management System* (WMS) : il est chargé d'accepter les travaux, de les envoyer vers le *Computing Element* (CE) approprié et d'enregistrer leur état et leur code de sortie auprès du système de *Logging and Bookkeeping*(LB) ;
- *Relation Grid Monitoring Architecture* (R-GMA) : il est une implantation du standard *Grid Monitoring Architecture* créé par le *Open Grid Forum* [35] et se présente comme une base de données relationnelle distribuée pour surveiller les éléments.
- *Local Resource Management System* (LRMS) : il est le gestionnaire local des ressources. Les différents types de LRMS (ou *batch scheduler*) reconnus sont OpenPBS [105], LSF, Maui/Torque [108] et Condor [91].
- *LCG File Catalog* : il permet de stocker une description et une localisation des données disponibles ;

Tous ces composants ou briques logicielles coordonnent et organisent les ressources de la grille EGEE dans le but de rendre possible l'utilisation de ces ressources par un grand nombre d'utilisateurs.

GLite n'est pas figé et des améliorations sont apportées aux éléments qui le façonnent au fur et à mesure de l'évolution du projet. La première version de l'intergiciel est apparue en 2005, il est actuellement dans sa version 3.1. Nous verrons dans le chapitre 3, section 3.3.5 un exemple de mise en œuvre.

Au mois de juillet 2008, le projet EGEE [76] dénombrait 259 sites répartis sur 59 pays. Il comptait environ 79 000 processeurs et 20 PetaBytes d'espace de stockage, plus de 7 500 utilisateurs enregistrés au travers de 130 organisations virtuelles et une activité d'environ 150 000 jobs/jour.

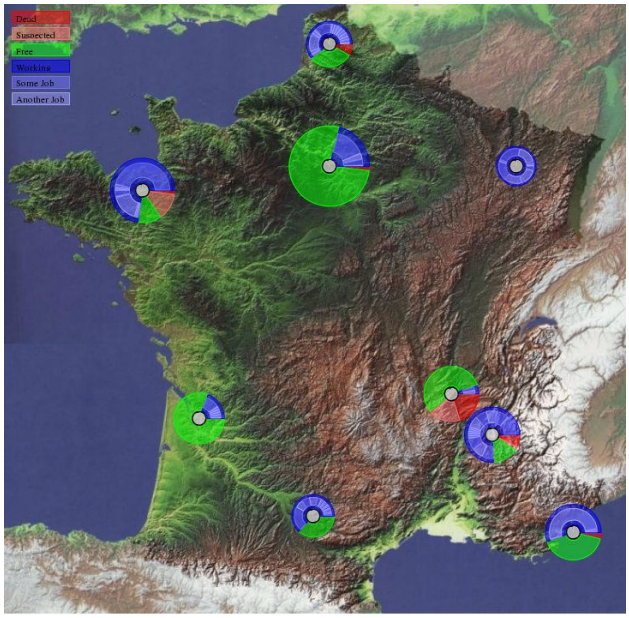


Figure 1.3 – Grid'5000 à travers la France.

1.3.2 Grid'5000

Le projet Grid'5000 [98] a démarré en 2003, il se poursuit avec l'Action de Développement Technologique INRIA ALADDIN-G5K. Il vise à mettre à la disposition des chercheurs une plate-forme expérimentale comme un outil de recherche et comme un environnement de tests pour les grilles informatiques.

Par nature, les grilles informatiques sont complexes à mettre en place de par la multitude de logiciels, protocoles et matériels sur lesquels elles reposent. Comme la plupart des scientifiques, les chercheurs en informatique ont besoin d'un environnement de validation. Aussi lorsque la recherche porte sur l'informatique distribuée, il est primordial d'avoir à disposition un instrument permettant d'avoir des conditions reproductibles, extensibles, réalistes et contrôlables.

La plate-forme Grid'5000 offre la possibilité de configurer entièrement toutes les piles de logiciels présentes sur les machines qui la composent. Elle permet de mettre en place un environnement contrôlé reproduisant les conditions d'utilisation des grilles informatiques. Ainsi, la démarche scientifique peut être appliquée dans le cadre de l'informatique distribuée à l'aide de cette plate-forme expérimentale. Grid'5000 est en ce sens, un outil d'avant garde pour l'informatique distribuée. Cette plate-forme ajoute une méthode supplémentaire de validation « in situ » d'algorithmes, de protocoles de com-

munication, d'implantations logiciels et d'applications.

La grille Grid'5000 est composée de 9 sites (Bordeaux, Grenoble, Nancy, Orsay, Lille, Lyon, Rennes, Sophia, Toulouse) répartis sur le territoire français (voir figure 1.3). Nous dénombrons plus de 580 utilisateurs supportés par un peu moins de 10 administrateurs à temps complet ou partiel. L'architecture de Grid'5000 repose sur une fédération de grappes de processeurs. Actuellement, la plate-forme compte un peu plus de 3400 processeurs qui sont majoritairement d'architecture x86-64 (AMD Opteron) ou EMT64 (Intel Xeon) et près de 4800 cœurs. Grid'5000 s'appuie sur le réseau universitaire RENATER [80] pour relier chaque site avec des connexions à 1 ou 10 Gb/s. À l'intérieur d'un site, les machines sont interconnectées par des réseaux Gigabit Ethernet, parfois Myrinet comme sur les sites de Bordeaux, Grenoble, Orsay, Lyon, Rennes et Sophia-Antipolis, ainsi qu'Infiniband, comme sur le site de Rennes.

Peut-on parler d'un intergiciel de grille pour Grid'5000 ?

La plate-forme Grid'5000 repose sur un ensemble de briques logicielles, protocoles standards, ressources délocalisées et une gestion non-centralisée. Elle offre des mécanismes de réservation et déploiement de machines, de la sécurité et un espace de stockage. D'après la définition que nous avons donnée, nous pouvons bien parler de grille informatique. Cependant il est impossible de parler d'un intergiciel en particulier pour Grid'5000, mais plutôt d'un ensemble de services disponibles :

- un espace de stockage propre à chaque site : NFS locaux ;
- une base de données des comptes utilisateurs décentralisée, répliquée : annuaire LDAP ;
- un gestionnaire local de réservation de ressources décentralisé sur chaque site : OAR [104] ;
- un réseau permettant la communication entre chaque machine de la grille ;
- un gestionnaire de déploiement et de démarrage : Kadeploy [103].

En effet, si nous ne parlons pas d'un intergiciel pour Grid'5000, c'est justement parce qu'un des buts de cette plate-forme est de permettre la conception, le test et le développement d'intergiciels de grilles informatiques. Mais il est vrai que, prises ensemble, toutes les briques logicielles de Grid'5000 forment un intergiciel. Nous pouvons citer le travail effectué autour d'outils comme GRUDU[1] qui fédère tous les services de Grid'5000 afin d'offrir un service d'ensemble sur tous les sites (voir la figure 1.4).

[1]GRUDU : ***G****rid'5000* ***R****eservation* ***U****tility for* ***D****eployment* ***U****sage*, développé dans l'équipe GRAAL à Lyon.

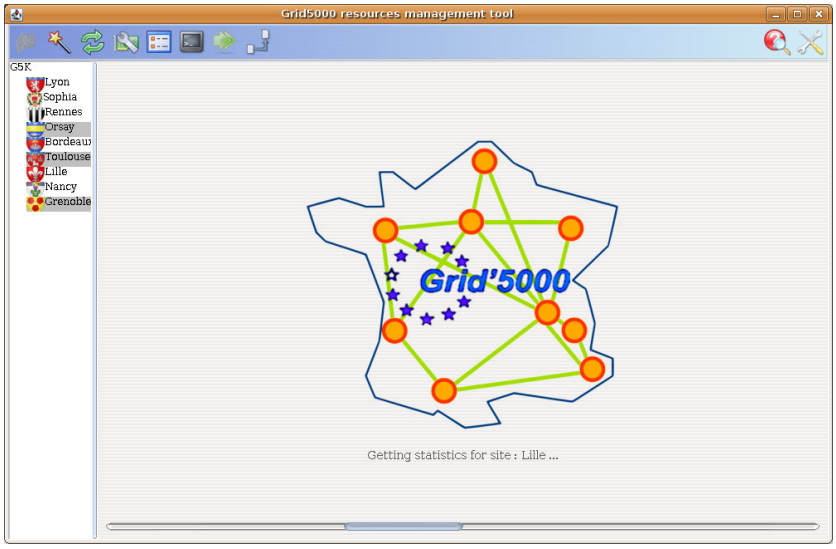


Figure 1.4 – Outils de réservation GRUDU.

Nous verrons dans ce manuscrit (chapitre 4), à travers plusieurs expériences, comment nous avons pu utiliser la grille de recherche Grid'5000 pour la validation des fonctionnalités de l'intergiciel DIET et préparer la phase calculatoire d'un projet scientifique.

1.3.3 World Community Grid

Le World Community Grid [101] est un projet initié et supporté par IBM et de nombreux partenaires. La mission visée par ce projet est de devenir la plus vaste grille de calcul distribuée au monde dans le but d'aider des projets scientifiques qui profiteront l'humanité toute entière.

Derrière cet objectif ambitieux et philanthropique, les différents acteurs de ce projet veulent démontrer leur savoir faire et leur capacité à relever les défis techniques liés à ce challenge. Ils montrent aussi leur engagement dans des projets médiatiques qui leur permettent de cultiver une image positive et novatrice.

Outre l'aspect purement médiatique la grille du World Community Grid est un exemple d'une grille informatique dite de « volontaires ». Elle s'appuie sur les ressources informatiques (ordinateurs personnels et connexion internet) de ses 400 000 membres pour réaliser des calculs.

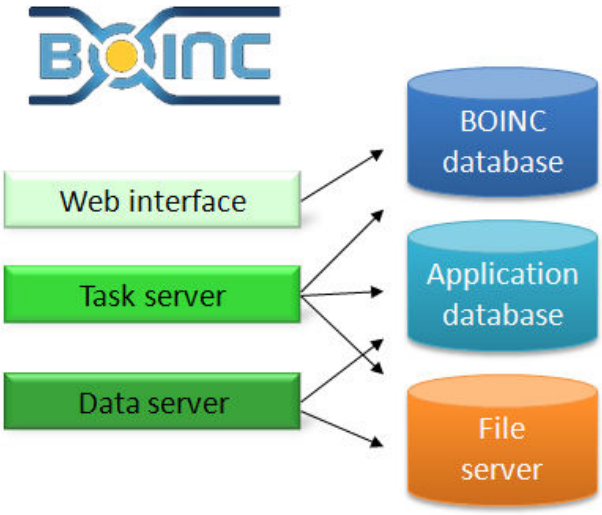


Figure 1.5 – les différents éléments de BOINC.

BOINC un intergiciel de grille

Techniquement, le World Community Grid s'appuie sur l'intergiciel BOINC (*Berkeley Open Infrastructure for Network Computing*) [7, 90] pour assurer la distribution des calculs.

Les volontaires inscrits sur le site du World Community Grid installent un logiciel client qui a pour fonction de télécharger les calculs à effectuer. Le logiciel client est responsable du lancement et du déroulement des calculs sur la ressource informatique du volontaire. L'utilisateur de la machine reste maître de son ordinateur et il peut à tout moment arrêter un calcul, ou un transfert en cours. Le logiciel client BOINC est entièrement paramétrable pour effectuer des calculs en continu, ou uniquement durant certaines périodes de la journée, ou bien encore lorsque la machine n'est pas utilisée.

Dans l'architecture présentée dans la figure 1.5, les clients (volontaires) ne sont pas des clients au sens commun, c'est à dire les utilisateurs des ressources de la grille. Ils sont les acteurs, les moyens de calcul de la grille.

Plusieurs adjectifs sont employés pour qualifier ce type de grille :

- grille de volontaires : les ressources de ces grilles sont fournies sur la base du volontariat.
- grille d'ordinateurs personnels : les volontaires participant à ces grilles le font avec leur ordinateur personnel ou professionnel. Cependant rien n'empêche d'inscrire des machines dédiées dans cette architecture.
- grille à vol de cycle : l'idée fondatrice de cette grille est basée sur le

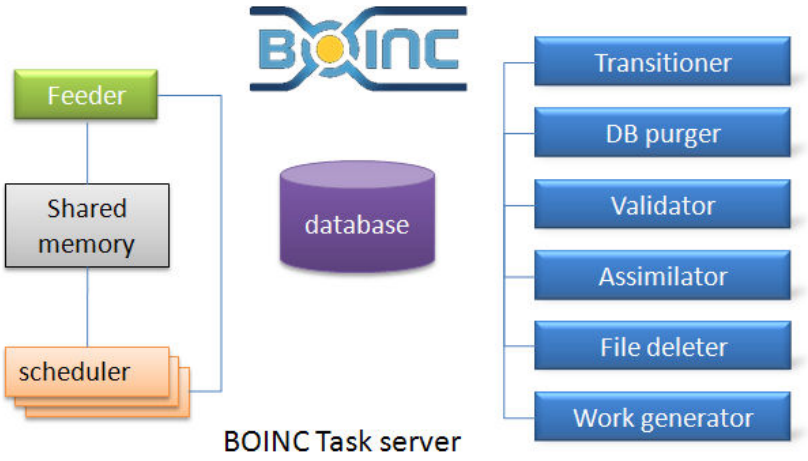


Figure 1.6 – Le serveur de tâches de BOINC.

constat que les ordinateurs utilisés interactivement le sont en réalité très peu en pourcentage (10 à 20 % du temps). L'idée est donc de *voler* les cycles d'horloge non utilisés. Cependant, les utilisateurs étant des volontaires, il est préférable de parler de « *don de cycles* ».

– grille volatile : étant donné le contrôle total laissé à l'utilisateur et propriétaire, les ressources ne peuvent pas être considérées comme stables et immuables.

L'architecture du serveur de tâches BOINC

Parmi les composants présentés dans la figure 1.6, le *work generator* crée de nouvelles tâches ainsi que les fichiers de paramètres correspondants. Il interrompt la création de tâches lorsque le nombre de tâches en attente dépasse un certain seuil. Le *scheduler* gère les demandes des clients. À chaque requête, le client demande du travail supplémentaire tout en indiquant ce qu'il a déjà fait. Le *scheduler* donne du travail au client en fonction de ces caractéristiques (système d'exploitation, quantité de mémoire, type de processeurs, *etc*). Le *feeder* a pour rôle de préparer des tâches pour le *scheduler* afin qu'il les dissémine aux clients. Les autres modules (*transitioner*, *validator*, *assimilator*, *file deleter* et *database purger*) ont pour rôle de traiter les résultats apportés par les clients. Ils garantissent une tolérance aux fautes. Chaque tâche est soumise suivant un taux de redondance. Les résultats obtenus sont comparés en cherchant à atteindre un quorum de résultats équivalents. Tant que ce quorum n'est pas atteint, la tâche continue

à être soumise sur d'autres machines.

Le projet World Community Grid compte actuellement plus de 400 000 membres volontaires et plus de 1 000 000 de logiciels clients potentiels. Nous verrons, dans le chapitre 4, une description des ressources du World Community Grid à base de *processeurs virtuels à plein temps* que nous introduirons.

1.4 Conclusion

Nous avons présenté un panorama de la notion de grille informatique. Nous avons dégagé quelques notions clefs qui composent une grille : les applications, l'intergiciel, les services et les ressources. Il existe une multitude de grilles différentes utilisant des intergiciels adaptés à chaque projet.

Nous avons décrit trois exemples qui représentent une illustration de projet de grille dans le paysage mondial, il en existe des centaines d'autres. Nous verrons dans le chapitre 3 comment nous avons mis en production une grille basée sur les ressources d'universités partenaires pour le projet Décrypthon et dans le chapitre 4 comment nous avons utilisé trois grilles complémentaires : la grille de recherche Grid'5000, le World Community Grid et la grille Décrypthon, pour entreprendre une phase de calcul qui aurait demandé près de 80 siècles sur une seule machine.

La grille est aujourd'hui une réalité et un domaine de recherche actif, nous n'avons pas encore atteint la métaphore du réseau électrique, où un utilisateur pourrait se connecter de manière transparente à un immense réservoir de ressources informatiques, cependant le rêve est en marche.

Chapitre 2

Le programme Décrypthon

Sommaire

2.1 Introduction

Le programme Décrython est un projet mené de front par trois acteurs. Il a pour but de financer chaque année des projets scientifiques qui ont besoin d'une importante quantité de calcul et/ou d'une importante ressource de stockage. Il vise un objectif : aider la recherche sur les maladies neuromusculaires et les maladies rares pour la plupart d'origine génétique. Nous verrons l'historique de ce programme, de sa création à son application, ainsi que l'ensemble des projets sélectionnés.

2.2 Le programme Décrypthon

2.2.1 Historique

En 2001, lors du très médiatique Téléthon, une première opération « Décrypthon » avait permis la réalisation d'une première base de données listant toutes les protéines du monde vivant. Ce projet avait pour objectif de « cartographier » le protéome[1]. Entre décembre 2001 et mai 2002, 75 000 internautes avaient permis la comparaison, au moyen de l'algorithme de Smith-Waterman [88], de 560 000 protéines connues, provenant de diverses espèces, depuis des organismes unicellulaires comme les bactéries jusqu'aux vertébrés (dont la souris et l'homme), en passant par des organismes pluricellulaires tels que la drosophile (la mouche du vinaigre), le nématode (un ver), ou une plante (l'arabète). Ces travaux ont utilisé les séquences protéiques répertoriées dans plusieurs bases de données parmi lesquelles Swiss-prot, IPI, TrEMBL, et Refseq. Les calculs comparaient les séquences de ces protéines pour les classer en familles de protéines homologues à travers les différentes espèces. La base de données obtenue permettait la mise en relation de cette classification avec les structures tridimensionnelles connues de certaines de ces protéines. Pour résumer, il s'agissait d'une véritable cartographie du protéome. La base de données établie, d'une ampleur sans précédent, permettait aux chercheurs de travailler plus facilement et plus rapidement sur ces éléments complexes et essentiels de notre organisme que sont les protéines.

2.2.2 Naissance du programme

Forte de cette première expérience, l'AFM a désiré continuer l'initiative en démarrant une nouvelle coopération originale entre trois acteurs majeurs aux missions, aux domaines de compétences et aux stratégies distinctes :

[1] Ensemble des protéines synthétisées par une cellule.

– l'Association Française contre les myopathies (AFM) ;
– le Centre National de Recherche Scientifique (CNRS) ;
– la société Intelligent Business Machine (IBM France).

De cette collaboration tripartite est née en 2004 le programme Décrypthon. Il s'inscrit sur des objectifs à long terme.

Ainsi L'AFM coordonne tous les ans un appel à projets auprès de la communauté scientifique, et contribue au financement de ces projets sous plusieurs formes : financement direct aux équipes et financement d'éventuels services associés. Ce programme s'inscrit dans la stratégie scientifique de l'AFM qui vise un objectif : guérir les maladies neuromusculaires et les maladies rares pour la plupart d'origine génétique.

IBM apporte son expertise technique dans les technologies de grilles et son expérience dans la conduite de projets. De plus, dans le cadre du programme IBM *Shared University Research*, IBM France dote certaines universités de supercalculateurs de dernière génération.

Le CNRS assure le pilotage du programme scientifique. Il apporte également une expertise scientifique et technologique sur le portage des projets sur la grille et sur l'optimisation des moyens de calcul.

En complément de ces trois partenaires fondateurs, le programme s'appuie sur la participation de six universités (Bordeaux I, Lille I, ENS-Lyon, Paris IV, Pierre et Marie Curie et Orsay) où des supercalculateurs ont été installés par IBM, ainsi que sur le réseau à très haut débit RENATER (Réseau National de Télécommunications pour l'Enseignement et la Recherche), sur lequel est connecté l'ensemble de ces ressources. Le Centre de ressources informatiques de Haute Normandie (CRIHAN) participe également au programme, il héberge les données volumineuses des projets scientifiques.

L'objectif du programme Décrypthon est donc d'accompagner des projets scientifiques et de mettre à disposition une plate-forme de calcul et de stockage stable permettant d'accélérer la recherche en génomique et en protéomique. Nous détaillerons dans le chapitre 3 l'architecture de la grille Décrypthon.

2.2.3 Organisation du programme Décrypthon

Le programme Décrypthon est organisé en trois comités (voir figure 2.1) ayant chacun des responsabilités distinctes :

– le comité directeur : les trois partenaires fondateurs du programme se réunissent chaque mois. Ce comité est chargé du pilotage global. Il prend les décisions concernant le programme Décrypthon suivant les éclairages donnés par les deux autres comités ;

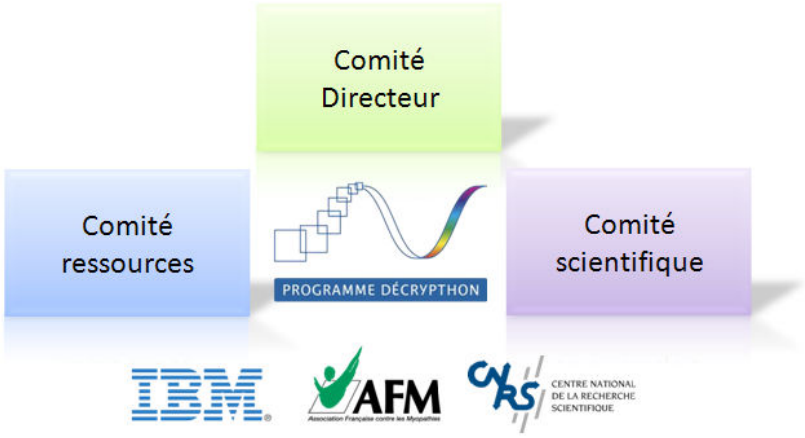


Figure 2.1 – L'organisation du programme Décrypthon.

- le comité scientifique : il est chargé de l'évaluation des projets scientifiques. Il juge leur pertinence et leur faisabilité dans le cadre fixé par le programme Décrypthon ;
- Le comité ressources : il est chargé de la gestion des ressources du programme Décrypthon, que ce soit au niveau des machines impliquées dans la grille Décrypthon ou au niveau des relations avec les universités partenaires du projet. Le personnel du CNRS et d'IBM pilote ce comité.

Le programme s'organise autour de projets qui ont été sélectionnés par une commission de scientifiques. Les équipes des projets sont alors prises en charge dans le programme Décrypthon afin qu'elles aient accès aux ressources informatiques que nous présenterons dans le chapitre suivant.

L'originalité du programme Décrypthon est de réunir les compétences d'équipes à la pointe dans le domaine des sciences du vivant afin de faire progresser les connaissances. La volonté affichée de ce programme est d'ouvrir les perspectives des grilles informatiques à des projets qui n'ont pas encore accès à cette technologie. Il existe déjà des initiatives similaires, citons par exemple le projet européen EGEE [76] et l'organisation virtuelle (VO) Biomed, à une échelle européenne.

2.3 Les projets scientifiques

Ces projets répondent à un appel d'offre publié tous les ans par l'AFM. Une fois le processus de sélection passé, les projets scientifiques se voient attribuer un financement. Ils sont suivis pendant une période de 18 mois par

le programme Décrypthon.

N'importe quel projet de recherche en biologie qui demande une importante quantité de calcul et/ou une importante ressource de stockage peut répondre à l'appel d'offre. La priorité est donnée aux :

- projets qui combinent des compétences en biologie et bioinformatique ;
- projets ayant une application informatique validée sur une étude préliminaire ;
- projets de recherche ayant un thème directement ou indirectement lié aux maladies neuromusculaires.

Une préférence est donnée aux projets ambitieux, ayant un intérêt à utiliser les grilles informatiques. Une attention particulière est donnée à la diffusion des résultats à l'issue du projet : ils devront, à terme, être accessibles à la communauté scientifique.

Dans la suite du document, nous présentons les projets qui ont été sélectionnés pour intégrer le programme Décrypthon durant la thèse. La description que nous faisons est un condensé du dossier scientifique des projets. Il donne une idée des thèmes de recherche abordés et des méthodes mises en œuvre. Quelques uns de ces projets sont en cours, et d'autres sont maintenant terminés. Enfin, d'autres sont encore en développement.

2.3.1 Le projet MS2PH

Équipes scientifiques impliquées

Ce projet est porté par 2 équipes de recherche situées à Lyon et à Strasbourg :

- l'équipe Bioinformatique intégrative et génomique,UMR 7104,IGBMC Illkirch, Olivier Poch ;
- l'équipe Bioinformatique et RMN structurales, UMR 5086, IBCP Lyon, Gilbert Deléage.

D'autres personnes sont impliquées directement ou indirectement dans ce projet : Luc Moulinier, Anne Friedrich, Julie Thompson-Maaloum, Ngoc-Hoan Nguyen, Emmanuel Bettler et Nicolas Garnier.

Objectifs

Le projet MS2PH est un acronyme pour « *de la **M**utation **S**tructurale aux **P**hénotypes des Pathologies **H**umaines* ». Ce projet poursuit le développement d'outils bioinformatiques et la mise à disposition de sources d'informations génomiques. Celles-ci permettent de faciliter la compréhension des relations qui existent au niveau des protéines impliquées dans les maladies

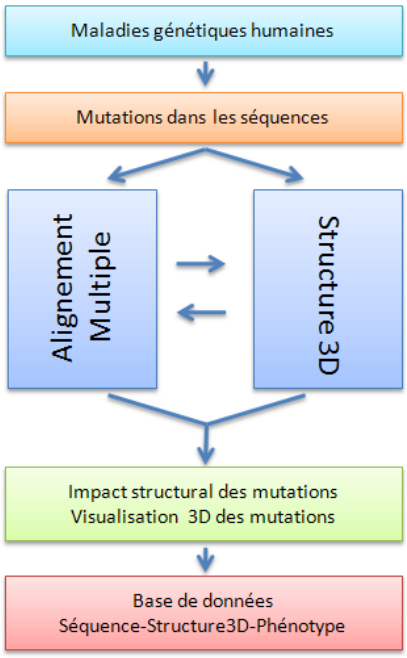


Figure 2.2 – Vue d'ensemble du projet MS2PH.

génétiques humaines. Les équipes s'intéressent plus particulièrement aux relations entre la séquence de la protéine, son évolution, sa structure tridimensionnelle et les pathologies associées. Le projet repose sur la longue expertise des équipes de bioinformatique de Strasbourg (IGBMC) et de Lyon (IBCP), et sur les nombreux outils et logiciels qu'ils ont développés. Ces programmes associent des phases de recherche dans des banques de données et des phases d'analyse afin de réaliser automatiquement :

- l'identification et la collecte d'homologues dans les banques de séquences, l'alignement de qualité de nombreuses séquences complètes de protéines. Il s'en suit une analyse par classification en sous-groupes des familles de protéines produites et une étude des conservations dans l'alignement ;
- la prédiction de structures secondaires sur la base des alignements, l'analyse de structures tridimensionnelles connues, la recherche de structures de même topologie, la modélisation moléculaire de protéines en couplant les données structurales d'homologues et celles déduites des familles de séquences.

À partir des données générées, il sera proposé aux biologistes un espace de travail réunissant des données structurales et évolutives, associées à des

données phénotypiques et mutationnelles de protéines impliquées dans des pathologies génétiques humaines. Le projet est ambitieux et s'inscrit dans le contexte fondamental de la compréhension des mécanismes qui contrôlent les fonctions des protéines.

Le projet est maintenant terminé. Nous décrirons le portage de son application dans la grille Décrypthon dans le chapitre suivant. Les technologies de grilles apportées par le programme Décrypthon ont permis de déporter une partie des calculs effectués localement sur les ressources du laboratoire IGBMC. Avec cette opportunité, les chercheurs des équipes impliquées ont pu envisager de traiter un plus grand nombre de séquences protéiques et de diviser par 30 les temps de calcul qu'ils pouvaient observer sur leurs propres ressources. De plus, une base de données relationnelle (MS2PH-db) a été créée. Elle contient des données structurales et évolutives, associées à des données phénotypiques et mutationnelles de 1036 protéines impliquées dans des pathologies génétiques humaines. Cette base de données est mise à jour automatiquement (bimensuellement) via les ressources Décrypthon. Un serveur Web, MAGOS [38], est également disponible. Il permet de réaliser une recherche exhaustive des séquences/structures homologues à une protéine cible impliquée dans une pathologie humaine. Il est couplé à la génération automatique d'une modélisation en 3 dimensions de la séquence protéique, basée sur la plus proche structure connue.

2.3.2 Défauts d'épissage et maladies génétiques

Équipes scientifiques et partenaires

- Le laboratoire Maturation des ARN et Enzymologie Moléculaire, UMR 7567, Nancy, Christaine Branlant ;
- l'équipe MODBIO, UMR 7567, LORIA, Nancy, Yann Guermeur ;
- le laboratoire Maturation des ARN et Enzymologie Moléculaire, UMR 7567, Nancy, Fabrice Leclerc.

Sont intervenus dans ce projet plusieurs collaborateurs financés par le programme Décrypthon : Petar Mitrasinovic, Nathalie Marmier-Gourrier, Delphine Autard, Emmanuel Monfrini, Juan Alexander, Padron García.

Objectifs

Le contexte : Pour synthétiser une protéine, le gène (présent dans la molécule d'ADN) délivre dans la cellule un code de fabrication dans le cadre d'un processus qui a été baptisé *épissage*. Au cours de cette étape, la cellule assemble bout à bout les éléments composant le code originel qui sont ap-

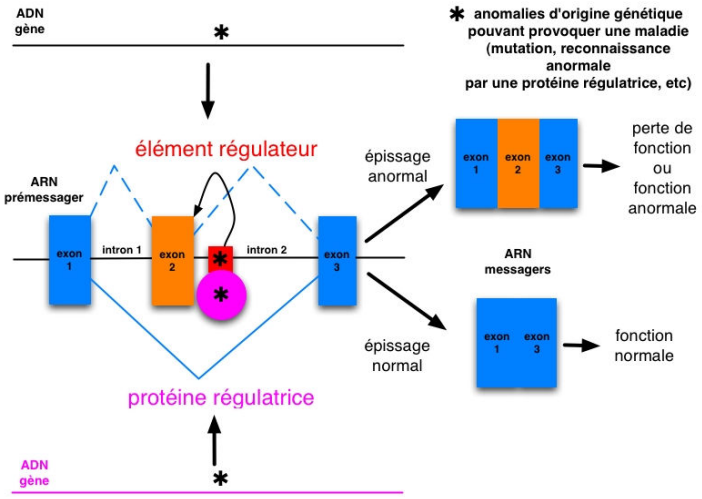


Figure 2.3 – Mécanisme d'épissage.

pelés exons. Le même gène peut donner lieu à plusieurs codes de fabrication différents, à la suite d'un épissage dit « alternatif ». C'est ce processus, représenté dans la figure 2.3, qui permet à un seul gène de produire plusieurs protéines à la fois.

L'analyse des mutations des gènes chez des personnes atteintes de maladies génétiques et les conséquences de l'épissage devraient apporter des données fondamentales pour la compréhension des maladies génétiques. Cette étude devrait permettre le développement de thérapies basées sur la réparation de l'épissage, impliquant, par exemple, le saut d'exon.

L'objectif du projet est d'analyser les relations existantes entre défauts d'épissage et maladies génétiques. Il s'intéresse aux déterminants de séquences (structure primaire) et de structures 2D et 3D des éléments des ARNm impliqués dans certains dysfonctionnements de l'épissage. L'objectif est double. Le premier est de développer des méthodes permettant d'identifier les sites d'épissage (par apprentissage statistique) ou de façon plus spécifique de distinguer les introns des exons. Le second est de développer des méthodes de modélisation moléculaire permettant de comprendre les bases moléculaires de la formation de complexes ARN/protéine spécifiques conduisant à des défauts d'épissage. Une meilleure prédiction par informatique des séquences introniques et exoniques dans les séquences génomiques humaines devrait permettre de prédire si des mutations identifiées sont susceptibles de conduire à un défaut d'épissage. La modélisation des complexes formés entre

les séquences répétées CUG/CCUG et les protéines régulatrices de l'épissage devrait permettre de comprendre les bases moléculaires des dystrophies myotoniques.

À long terme, ces développements et leurs applications à des maladies génétiques pourraient aboutir à des applications dans le domaine du diagnostique et du dépistage de maladies génétiques liées à des défauts d'épissage, et dans la conception de médicaments permettant d'atténuer les symptômes des dystrophies myotoniques.

Les travaux à réaliser seront :

- l'évaluation et le choix des données d'épissage ;
- la création d'une banque de données locale obtenue par extraction et annotation des données sur des sites d'épissage et leurs séquences environnantes ;
- le développement d'un programme de préparation des données pour l'apprentissage ;
- la mise en œuvre de la méthode d'apprentissage (tests et portage) destinée à l'identification introns/exons.

2.3.3 Help Cure Muscular Dystrophy

Équipes scientifiques et partenaires

- La faculté de médecine de la Pitié-Salpêtrière, Inserm U511 Immunologie cellulaire et moléculaire des infections parasitaires – Génomique analytique, Paris, Alessandra Carbone ;
- l'équipe PEQUAN,UMR 7606 LIP6, Paris Jean-Marie Chesneaux ;
- l'institut de myologie, UMRS 582, Paris, Pascale Guicheney ;
- le laboratoire de biochimie théorique, UPR 9080 Institut de Biologie Physico-Chimique, Paris, Richard Lavery.

Deux autres personnes ont participé directement à ce projet : Sophie Sacquin-Mora, Jean-Luc Lamotte.

Objectifs

Ce projet s'intéresse à la détection des sites d'interaction protéine-protéine, protéine-ADN et protéine-ligand. La méthode utilisée est celle du *docking* moléculaire. Cette dernière consiste à utiliser deux protéines « observées » en trois dimensions. L'une est fixe, tandis que la seconde est déplacée afin de déterminer les positions dans lesquelles elle « interagit » correctement avec sa consœur (figure 2.4). L'algorithme utilise un modèle réduit des macromolécules, associé à un champ de force simplifié. Les géométries d'interaction

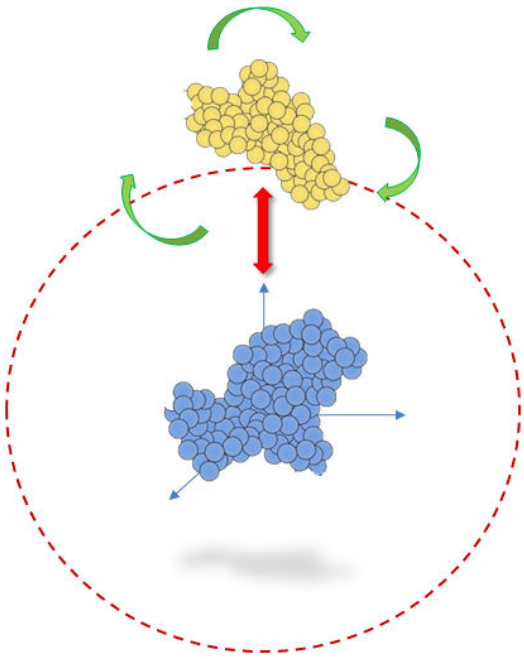


Figure 2.4 – Amarrage moléculaire ou *molecular docking*.

optimales sont localisées grâce à des minimisations d'énergies multiples, en partant d'un réseau régulier des points de départ et d'orientation.

Le projet se déroule en plusieurs étapes :

- une première étape de calcul sur une base de données de référence comportant 168 protéines sera lancée afin d'obtenir les données d'interaction.
- les données issues de la première étape de calcul permettront de valider la mise au point d'un algorithme de détection phylogénétique de sites d'interactions (JET) et ainsi permettre de réduire le nombre de points à explorer pendant les calculs d'amarrage.
- une deuxième étape de calcul sera lancée sur une base de données d'environ 4000 protéines collectées auprès de biologistes. L'amarrage moléculaire prendra en compte les informations issues de JET pour réduire l'espace d'exploration.

Identifier des paires ou des complexes de protéines interagissant ou déterminer le lien entre une protéine et un ligand sont des problèmes fondamentaux en biologie. Ce projet s'attaque directement à ces questions. L'objectif est de construire une base de données de plusieurs milliers de protéines. Leurs sites d'interaction avec d'autres protéines, ADN ou ligands seront prédits, caractérisés et comparés entre eux afin d'identifier les partenaires fonctionnels.

Les informations concernant la structure de complexes macromoléculaires sont importantes non seulement pour identifier les partenaires fonctionnels, mais également pour déterminer comment des mutations artificielles ou naturelles au sein du site affecteront les interactions avec les partenaires et avec des molécules exogènes telles que les pharmacophores. Une base de données contenant ce type d'informations serait d'un intérêt médical incontestable. Il est maintenant possible de décrire une molécule pour empêcher ou amplifier le lien d'une macromolécule donnée avec un partenaire spécifique. Cependant il est beaucoup plus difficile de savoir comment cette petite molécule peut directement ou indirectement influencer d'autres interactions existantes. Les impacts peuvent être immédiats notamment en pharmacologie.

Nous décrirons en détail l'application de ce projet dans les chapitres 3 et 4. Nous verrons comment ce projet a pu tirer pleinement profit de plusieurs grilles. Nous utiliserons à la fois les ressources Décrypthon pour le travail de mise au point et le paramétrage de leur application d'amarrage, et la grille de volontaires World Community Grid pour la réalisation de la phase I du projet. Celle-ci a demandé au total plus de 80 siècles de temps processeurs. La deuxième phase de calcul est actuellement en cours de préparation pour un nouveau lancement sur les ressources du World Community Grid.

2.3.4 SpikeOmatic : tri de potentiel d'action

Équipes scientifiques et partenaires

- Le laboratoire de physiologie cérébrale, CNRS UMR 8118, Christophe Pouzat ;
- le laboratoire de Physique Théorique de la Matière Condensée,CNRS UMR 7600, Pascal Viot.

D'autres personnes ont participé à ce projet directement ou indirectement : Matthieu Delescluse, Emmanuel Fournier, Jean Diebolt.

Objectifs

Les enregistrements extracellulaires de l'activité des neurones au moyen d'électrodes insérées dans le cerveau constituent une des techniques principales employées par les neurophysiologistes pour étudier le système nerveux central. Les neurologues, quant à eux, utilisent le même procédé pour diagnostiquer les maladies neuromusculaires en enregistrant l'activité des fibres du muscle. Ces enregistrements (figure[2] 2.5) sont constitués de la superposition de l'activité de plusieurs neurones ou fibres musculaires.

[2]extrait du manuel d'utilisation de spikeOmatic

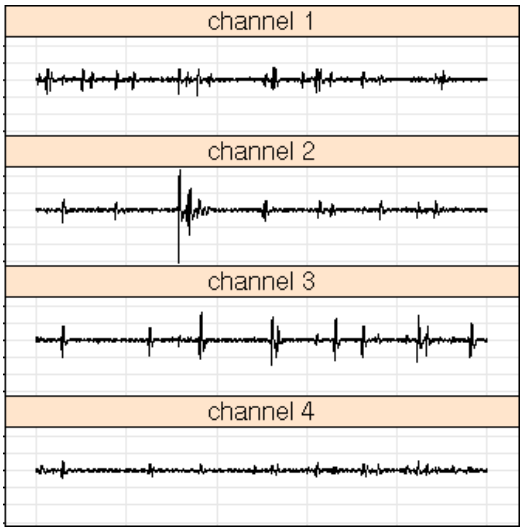


Figure 2.5 – Un exemple d'enregistrement de train d'onde.

La première étape de l'analyse des enregistrements est la séparation du « mélange » qui constitue le signal en ses différentes composantes. En effet, les potentiels d'actions sont émis par plusieurs neurones individuels. Cette séparation est le plus souvent faite manuellement par le neurophysiologiste. Non seulement ce mode de séparation demande du temps mais il introduit potentiellement des erreurs. Ainsi deux personnes analysant les mêmes données n'aboutiront pas systématiquement au même résultat, c'est-à-dire, au même diagnostic.

La technique d'enregistrement extracellulaire étant ancienne, le problème du « tri des potentiels d'actions » (qu'ils soient neuronaux ou musculaires) a été reconnu comme tel depuis longtemps, et de nombreuses méthodes automatiques ont été proposées. Néanmoins, jusqu'à très récemment, toutes ces méthodes étaient essentiellement basées sur l'amplitude des signaux enregistrés. Elles n'étaient capables de traiter les propriétés temporelles du signal que de façon approximative en modélisant la statistique de décharge des neurones par un processus de Poisson. De même, elles ne pouvaient rendre compte de façon satisfaisante d'une propriété fréquente des potentiels d'action : la décroissance de leur amplitude pendant les périodes d'activité « soutenue ». Les limites de ces méthodes automatiques ont conduit certains scientifiques à développer une nouvelle méthode construite sur un modèle probabiliste explicite. Celle-ci permet de déterminer correctement l'origine des potentiels d'actions générés par des neurones déchargeant en bouffées, cas où les méthodes précédentes échouent le plus souvent. Le prix à payer pour cette

meilleure fiabilité est l'utilisation de méthodes numériques intensives [75].

Le projet aborde aussi le délicat problème de la détermination automatique du nombre de neurones (ou unités motrices) présents dans le signal. Actuellement, l'utilisateur doit analyser ses données en utilisant différents modèles caractérisés par un nombre différent de neurones. Sur la base des résultats fournis par les modèles, il doit en choisir le meilleur. Il est possible de procéder de manière automatique et surtout plus rigoureuse, sur la base d'un critère d'information faisant intervenir la probabilité a posteriori des données conditionnées à un modèle. À nouveau, cela va entraîner la répétition d'un même type de calcul avec des modèles comportant différents nombres de neurones.

Enfin, le projet aboutira à une méthode fiable et efficace pour le « tri des potentiels d'actions » et répondra à un besoin important dans la communauté des neurophysiologistes [17]. À terme, un portail Internet sera accessible aux neurologues qui pourront y envoyer leurs électromyogrammes et obtenir en retour les mesures des potentiels d'action. Ces mesures, une fois interprétées, permettent d'établir un éventuel diagnostic de maladies neuromusculaires.

L'utilisation des technologies de grilles pour ce projet se justifie par l'utilisation déportée d'un service de tri des potentiels d'actions qu'il serait trop coûteux d'exécuter sur une seule ressource locale. L'application informatique de ce projet et sa mise en œuvre seront décrites dans le chapitre suivant.

2.3.5 Expression de l'évolution des gènes : GEE

Équipes scientifiques et partenaires

- Le laboratoire de Biologie Moléculaire de la Cellule, UMR 5161, ENS-Lyon, Marc Robinson-Rechavi ;
- l'équipe GRAAL,(LIP - ENS-Lyon - CNRS - INRIA), Raphaël Bolze, Frédéric Desprez ;
- l'École Normale Supérieure de Lyon, UMR 5161 CNRS INRA, Laurent Schaeffer.

Objectifs

La transcriptomique ouvre l'accès à des données d'expression à grande échelle, qui contiennent de l'information sur la fonction des gènes dans le génome. Mais en raison du bruit dans les données, et de la nature stochastique de l'évolution de l'expression, le passage du transcriptome à une information médicalement pertinente sur les gènes se révèle difficile.

Dans ce projet, les gènes humains sont annotés en utilisant non seulement leur patron d'expression dans des tissus neurologiques ou musculaires, mais aussi l'expression de leurs homologues dans d'autres espèces. Ce projet permet de proposer de nouvelles cibles pour de futures études, par l'annotation du génome humain par rapport à leur implication dans des processus neuromusculaires normaux ou pathologiques.

Les objectifs scientifiques visent la mise au point d'une chaîne d'annotations efficace. Elle est basée sur l'analyse comparative de l'expression des gènes, et la mise en évidence de la conservation ou l'évolution de l'expression des gènes chez les animaux, pour un processus biologique bien défini (le développement et le fonctionnement neuromusculaire). La méthodologie utilisée comprend les étapes suivantes : attribution des données d'expression aux gènes animaux par similarité de séquences ; intégration de l'information d'expression dans une ontologie inter-espèces ; regroupement des gènes en familles, avec les alignements et les phylogénies correspondantes ; sélection guidée des familles de gènes avec des patrons d'expression pertinents.

Les comparaisons de séquences, pour l'attribution de données d'expression ou pour le regroupement des gènes en familles, et la construction de phylogénies, nécessitent beaucoup de calcul. De même pour la classification et la sélection de patrons d'expression. L'application requiert, en outre, la manipulation d'une grande quantité de données lors de la confection des banques de données spécifiques à une espèce et la recherche des expressions de gène commune à ces espèces.

2.3.6 Échantillonnage conformationnel et docking

Équipes scientifiques et partenaires

- L'équipe OPAC-DOLPHIN (LIFL – CNRS – INRIA –USTL), El-Ghazali Talbi ;
- l'équipe INSERM U422, Nicolas Sergeant ;
- l'équipe IBL, D. Horvath, B. Parent.

Parmi ces équipes, voici la liste des personnes impliquées dans ce projet, en plus des responsables : Nordine Melab, Jourdan Laetitia, Alexandru Tantar, Jean-Charles Boisson.

Objectifs

La modélisation moléculaire, et notamment les procédures d'échantillonnage conformationnel et de « *docking* », sont des outils capables d'aider à la résolution des mécanismes d'interaction des (macro)molécules à la base des

processus physiologiques. Au-delà de l'intérêt évident de pouvoir prédire par calcul les modes de liaison des partenaires formant des complexes à rôle régulateur dans la cellule vivante, cette démarche peut également être utilisée pour la recherche « In Silico » (par calcul) des moyens d'interférer avec le processus physiologique normal ou pathologique.

Le groupe de D. Horvath à l'Institut de Biologie de Lille développe une méthodologie originale d'échantillonnage conformationnel. L'approche se base sur un algorithme génétique (AG) « codant » les conformations moléculaires sous la forme de « chromosomes » numériques (listes de valeurs d'angles torsionnels associées aux rotations autour des liaisons covalentes), ou l'énergie intramoléculaire joue le rôle de score de « *fitness* » gérant la sélection des chromosomes « les plus forts ».

L'originalité de cette démarche réside dans l'hybridation de l'approche avec des algorithmes génétiques et d'autres méthodes d'optimisation (déterministes et stochastiques) pour mieux s'adapter aux spécificités de l'échantillonnage conformationnel. En parallèle, l'approche sera généralisée pour inclure des degrés de liberté intermoléculaires, en devenant ainsi un outil de « *docking* » flexible.

Pour pouvoir répondre aux questions spécifiques sur l'interaction protéine/ARN dans les processus d'épissage qui interviennent dans les maladies neuromusculaires, une collaboration avec le projet *Défauts d'épissage et maladies génétiques* (paragraphe 2.3) permettra d'utiliser ce processus d'échantillonnage/docking pour la génération étendue d'hypothèses diverses d'interaction ARN/protéine. Elles seront évaluées ensuite selon les méthodes spécifiques pour acides nucléiques développées à Nancy. En effet, la forte charge portée par les ADN/ARN ne permet pas une évaluation de l'énergie par le champ de force simplifié de l'algorithme génétique, alors que l'approche Monte-Carlo utilisée à Nancy n'a pas une capacité suffisante d'échantillonnage : cette complémentarité est donc une forte incitation à collaborer.

En parallèle du développement des outils informatiques, le projet consistera à appliquer ces outils dans le domaine de la dystrophie myotonique pour laquelle des interactions protéine/ARN sont supposées jouer un rôle central dans l'étiologie liée à cette maladie neuromusculaire. Cette dernière partie devrait permettre de rechercher des analogues structuraux mais également d'orienter une recherche pharmacologique de petites molécules qui pourraient être étudiées dans un modèle cellulaire du laboratoire du D. Horvath.

Ce projet n'a pas besoin des ressources Décrypthon que nous décrirons dans le chapitre suivant. Il utilise ses propres infrastructures. Il a, comme le projet HCMD, un grand besoin en temps processeur, et il utilise un intergiciel spécialement implanté. Il permet un processus de « méta optimisation » qui échantillonne l'espace de ces paramètres à la recherche du paramétrage

optimal, en rendant les essais successifs d'échantillonnage conformationnel de plus en plus performants.

2.3.7 De la génomique fonctionnelle à la compréhension du muscle

Équipes scientifiques et partenaires

- l'université de Lausanne, Institut Suisse de Bioinformatique, Lausanne, Robinson-Rechavi Marc ;
- l'École Normale Supérieure de Lyon, UMR 5161 CNRS INRA, Laurent Schaeffer.

D'autres personnes seront amenées à travailler sur ce projet, Guillot Evelyne, Vidhya Jagannathan, Pilot Fanny, Gangloff Yann-Gaël, Parmentier Gilles, Mazelin Laetitia, Roux Julien, Morette Sébastien et Chopin Émilie.

Objectifs

Pendant la différenciation, les cellules musculaires et leurs précurseurs expriment des milliers de gènes. Ceux-ci agissent probablement en se combinant en un nombre limité de réseaux de signalisation ou de métabolisme, mais le nombre de ces réseaux peut encore être de quelques centaines. Pour comprendre le contrôle génétique et moléculaire de la myogenèse et de la physiologie musculaire, nous devons caractériser ces réseaux. De plus en plus, des expériences à haut débit sont utilisées pour identifier les intervenants, mais des approches informatiques sont nécessaires pour donner un sens en terme de fonction musculaire à des données hétérogènes et abondantes.

L'intérêt général de ce projet est donc de construire des réseaux de signalisation pour le fonctionnement normal et pathologique du muscle, à partir de données haut débit pertinentes.

Les objectifs scientifiques sont une meilleure description moléculaire du muscle et de ses pathologies, ainsi que l'identification et la caractérisation préliminaire de nouveaux gènes cibles pour les maladies musculaires.

La méthodologie inclut du développement informatique et expérimental. Les méthodes informatiques comprennent la construction d'une base de données, la classification de gènes selon leur patron d'expression, d'après des données de transcriptome diverses, et la fusion de données d'interactome de diverses origines. Les méthodes expérimentales comprennent des expériences de biopuces et d'interactome, guidées par les résultats informatiques, ainsi que des expériences de biologie cellulaire sur des gènes cibles.

La classification de gènes d'après de grandes quantités de données de transcriptome, et la construction de réseaux à partir de données de transcriptome et d'interactome, nécessiteront des milliards d'opérations, qui se prêtent bien à la distribution sur une grille de calcul.

Ce projet devrait fournir des informations sur de nouveaux mécanismes de contrôle du développement musculaire et la physiologie, liées à la voie PI3K-mTOR. L'intégration de données d'interactome avec celles d'études du phénome et du transcriptome dans le muscle, et avec des interactions déjà connues, devrait conduire à l'identification de nouveaux modulateurs ou partenaires de la voie PI3K-mTOR. Ce projet pourrait fournir des informations sur les mécanismes d'interaction entre PI3K et d'autres voies également impliquées dans la physiologie du muscle, ainsi que dans les processus de régulation de modifications post-traductionnelles, de *turnover*, de compartimentation subcellulaire, de régulation rétroactive, et de spécificité du contexte des réponses des voies de signalisation.

Ce projet s'incrit dans le prolongement du projet décrit paragraphe 2.3.5. Il est en cours d'étude. Il nécessitera aussi la manipulation d'une grande quantité de données.

2.3.8 Identification à grande échelle des réseaux transcriptionnels durant la myogénèse

Équipes scientifiques et partenaires

- l'équipe Bioinformatique intégrative et génomique,UMR 7104,IGBMC Illkirch, Olivier Poch ;
- l'équipe Réseaux moléculaires des cellules souches progénitrices du muscle in vivo, UMR-S 787 Groupe Myologie, INSERM - UPMC Paris VI, Frédéric Relaix.

Plusieurs membres des deux équipes mises en jeu dans le projet seront amenés à intervenir : Perrodou Emmanuel, Thompson Julie, Ripp Raymond, Alonso Martin, Rochat Anne, Aurade Frédéric.

Objectifs

Ce projet a pour but d'identifier les mécanismes de transcription moléculaire impliqués dans la progression myogénique « in vivo ».

Les cellules souches jouent un rôle essentiel dans le développement et l'entretien des organes et tissus du corps humain et animal. Au moment de la croissance ou de la régénération des tissus, les cellules musculaires du squelette sont incapables de se diviser mais se forment à partir d'une population

de cellules souches progénitrices, seules capables de se diviser, de produire des copies d'elles même ou encore de différencier les cellules musculaires. F. Relaix a identifié et caractérisé une nouvelle population de cellules souches progénitrices jouant un rôle essentiel dans la génération de la majeure partie des cellules musculaires du squelette, y compris la population de cellules souches myogéniques chez l'adulte, et identifié les régulateurs de transcription responsables de la spécification et de la prolifération de ces cellules.

La stratégie principale consiste au développement d'une étroite collaboration entre le groupe de F. Relaix, dans le laboratoire duquel toutes les données biologiques seront générées et validées, et l'équipe de O. Poch, qui possédera les outils et les compétences informatiques nécessaires au traitement de ces données. Le principal outil de recherche de ce projet sera la souris, seul système mammifère se prêtant à des analyses génétiques moléculaires exhaustives.

Le projet s'appuiera sur les développements précédents du projet décrit 2.3.1 niveau de la grille Décrypthon. De nouveaux protocoles seront développés incluant les recherches de type PSI-Blast afin d'identifier les molécules similaires mais aux caractéristiques éloignées. Des protocoles assurant le stockage et la mise à jour dans une base de données au sein de Décrypthon seront développés.

Les résultats seront combinés avec les données de l'analyse transcriptomale effectuée « in vivo ». Cette approche complémentaire permettra d'identifier et de caractériser les réseaux moléculaires impliqués dans le développement, la spécification, la différenciation et la régénération musculaire. La validation fonctionnelle sera effectuée en utilisant les outils de génétique murine et l'expertise en biologie du muscle dans le laboratoire de F. Relaix.

2.4 Conclusion

De tous ces projets scientifiques, il faut retenir qu'ils sont tous demandeurs soit d'une quantité importante de calcul, soit d'un grand volume de stockage, ou même des deux. Parmi ces projets, certains associent des équipes de biologistes, de bioinformaticiens et/ou d'informaticiens. Cependant, tous ces projets ont besoin d'un outil opérationnel pour atteindre les objectifs qu'ils se sont fixés.

Ils ont tous une vocation de recherche, cependant ce n'est ni la façon de réaliser les calculs, ni même les moyens mis en œuvre pour les obtenir qui est l'objet de leur recherche. Avant tout, les projets sont demandeurs de ressources informatiques additionnelles.

C'est dans ce contexte que le programme Décrypthon intervient en propo-

sant de mettre à disposition une grille informatique et une équipe d'experts capables de répondre aux attentes de ces équipes scientifiques en leur donnant les moyens d'obtenir leurs résultats. Nous allons décrire dans le chapitre suivant la naissance de la grille Décrypthon.

Chapitre 3

Genèse de la grille Décrypthon

Sommaire

3.1 Introduction

Le programme Décrypthon repose sur un objectif simple : utiliser et mettre à disposition les technologies des grilles de calcul afin d'accélérer les recherches scientifiques dans les domaines qui intéressent l'AFM. Le chapitre précédent présentait les projets scientifiques et l'organisation du programme Décrypthon. Fin 2004, le comité directeur a demandé à la société IBM, avec le concours du CNRS, de construire une grille de calcul. Celle-ci devait permettre de répondre aux besoins des projets scientifiques susceptibles de bénéficier de ces technologies.

Dans ce chapitre, nous présenterons l'ensemble des étapes qui ont jalonné la création de la grille Décrypthon. Une méthode d'évaluation d'intergiciels sera proposée dans le but de choisir l'intergiciel qui servira à la mise en production de celle-ci.

Nous décrirons ensuite la première version de la grille Décrypthon et son fonctionnement, puis la phase de transition entre la grille gérée par l'intergiciel de la société *United Device* et la réflexion pour adopter une solution basée sur du code libre.

Après avoir décrit les fonctionnalités de l'intergiciel DIET, nous détaillerons la nouvelle architecture de la grille Décrypthon basée sur celui-ci.

Pour finir, nous présenterons le tableau de bord (*DIET_Webboard*) qui orchestre les fonctionnalités d'une grille gérée par l'intergiciel DIET au travers d'un portail internet.

3.2 Évaluation des intergiciels de grille

En collaboration avec l'entité « Business Consulting Service » d'IBM, nous avons mené une étude comparative sur plusieurs solutions de gestion de grille. Celle-ci avait pour but de sélectionner un intergiciel pour la plate-forme Décrypthon.

La grille Décrypthon est destinée à être installée dans des centres de calcul universitaires désirant participer au projet. Le réseau universitaire RENATER fournit l'interconnexion entre les sites. Ainsi, chaque centre de calcul se voit doté par IBM d'une ou plusieurs machines parallèles qui s'intègrent dans leur parc informatique afin de constituer les ressources de calcul. Initialement voici les objectifs et contraintes qui encadraient notre évaluation ;

- fédération de plusieurs centres de calcul universitaires ;
- incorporation de nouvelles ressources propres au Décrypthon dans les centres de calcul ;
- intégration dans l'environnement existant : matériel et logiciel ;

- aucune perturbation pour les utilisateurs locaux des centres de calcul universitaires ;
- administration légère voir inexistante pour les centres de calcul hébergeant les ressources Décrypthon ;
- contraintes de sécurité fortes ;
- grille destinée à des projets scientifiques de nature différente : calcul intensif et/ou manipulation importante de données ;
- portail Web d'accès à la grille et/ou ligne de commande.
- grille opérationnelle à une échéance de 4 mois.

Il est important de noter que le délai de mise en œuvre de la grille Décrypthon était court. Au bout de ces 4 mois, il nous fallait mettre en place la grille et avoir porté au moins un des projets pilotes.

De plus, le délai accordé pour l'évaluation et le choix de l'intergiciel était très réduit (2 mois). Il comprenait la mise en place de la méthode (listing des axes, thèmes et critères), la hiérarchisation des thèmes et critères (système de pondération) et l'application de la méthode.

Nous n'avons donc pas pu faire l'ensemble des tests voulus, d'autant plus que nous n'avions pas accès à des plates-formes de tests pour évaluer correctement les solutions proposées. Notre évaluation se basait donc sur les informations mentionnées dans la documentation et sur les informations demandées aux éditeurs, avec tous les biais que cela pouvait introduire. Au moment de la création du programme un seul ingénieur, en charge de la mise en production et du portage des applications, était engagé sur ce projet. Nous ne pouvions pas prétendre faire des développements au niveau de l'intergiciel de la grille. Le travail essentiel devait se trouver dans l'aide au portage des applications dans un environnement grille.

Cette étude comparative a mis en concurrence des solutions commerciales pré-sélectionnées par l'équipe d'IBM parmi les partenaires de l'entreprise. Nous regrettons de ne pas avoir disposé du temps nécessaire pour faire cette étude en profondeur. Nous décrirons la méthodologie employée sans dévoiler précisément les résultats de cette étude pour des raisons de confidentialité. De plus, selon nous, même si elle est imparfaite, elle est intéressante et permet dans l'absolu de pouvoir juger sur un ensemble de critères objectifs. Ces critères sont ensuite pondérés en fonction des besoins exprimés au moment de l'évaluation.

Au final, la décision du choix de l'intergiciel de la grille Décrypthon a été prise par le comité directeur à la lumière de cette étude.

3.2.1 Méthodologie : axes d'analyse, thèmes et critères

Afin d'obtenir une comparaison entre les différentes solutions commerciales nous les avons évaluées suivant différents critères. L'évaluation s'articule sur trois niveaux : un axe d'analyse développé en thèmes, eux-mêmes composés de critères. Les critères retenus sont autant de questions posées dont les réponses nous permettent d'établir une note. Les questions posées ne sont pas une liste exhaustive de celles que l'on doit se poser, mais elles constituent néanmoins une série de points que nous avons jugée utile d'analyser.

Fonctionnalités

Ces fonctionnalités sont d'ores et déjà orientées par la vision et le contexte du programme. Nous avons défini un ensemble de critères qui sont autant de points techniques souhaitables dans une grille de calcul. Les critères sont groupés en 9 thèmes qui reflètent les points qui nécessitent d'être abordés. Certains critères se chevauchent et peuvent être classés dans plusieurs thèmes. Nous en proposons le découpage suivant :

- La sécurité :
    - Comment s'effectue l'identification et l'authentification de chaque utilisateur ?
    - Le cryptage des échanges de données lors des transferts est-il effectué ?
    - Existe-t-il un mécanisme assurant l'intégrité des données lors du transfert et du stockage ?
    - Les privilèges : existe-t-il différents niveaux de privilège pour l'accès aux ressources (calcul et données) ?
    - Existe-t-il des restrictions d'accés aux données pour les utilisateurs de la plate-forme ?
- La tolérance aux pannes
    - Niveau intergiciel : Comment est assuré l'intégrité et la stabilité de l'intergiciel lors du disfonctionnement d'un de ses composants ?
    - Niveau ressources : la grille reste-t-elle opérationnelle malgré la panne ou l'indisponibilité d'une de ses ressources ?
    - Redondance : Est-il possible d'installer plusieurs entités d'un même composant pour assurer la tolérance aux pannes ?
- Passage à l'échelle
    - Quelle est la quantité de ressources (nombre de machine, espace de stockage, *etc*) que l'intergiciel peut gérer ?
    - Quel est le nombre d'utilisateurs qu'il peut accueillir ?

    - Combien de tâches peut-il contrôler en parallèle ?
- Gestion des ressources de calcul
    - Déclaration d'une nouvelle ressource : la déclaration d'une nouvelle ressource engendre-t-elle l'indisponibilité de la plate-forme ?
    - Allocation d'une ressource : est-il possible d'allouer une ressource à un utilisateur ou à un groupe d'utilisateurs ?
    - Partage des ressources : les ressources peuvent-elles être partagées entre les utilisateurs de la plate-forme et les utilisateurs locaux ?
- Gestion des données
    - Réplication : est-il prévu de répliquer une donnée dans le système et de la mettre à jour ?
    - Cohérence des données : existe-t-il une gestion des données répliquées et modifiables ?
    - La gestion des données est-elle ouverte vers d'autres systèmes : GPFS, Avaki, GridFTP, ... ;
- Ordonnancement
    - Quel est le mécanisme de sélection d'une ressource ?
    - Quels sont les paramètres utilisés pour l'ordonnancement ?
    - Peut-on définir des priorités pour les tâches ?
    - Mise en attente d'une tâche : Peut-on arrêter les calculs effectués sur une tâche ?
    - Synchrone/asynchrone : Quel est le mode de lancement (interactif/*batch*) des tâches ?
    - Ordonnancement spécifique : Est-il possible de définir un ordonnancement spécifique pour une application, un utilisateur ?
- Information et métrologie
    - Des services de l'intergiciel : détails de chaque service de la plate-forme ;
    - Des ressources de la plate-forme ;
    - État du système : rapport dynamique de l'état de la plate-forme ;
    - Statistique sur les services : rapport sur l'utilisation des services ;
    - Statistique sur les ressources : données et travaux ;
    - Statistique des travaux dans le système ;
    - Détection de pannes : alerte sur la détection de pannes ou d'anomalies ;
- Environnement d'installation
    - Systèmes d'exploitation supportés par l'intergiciel ;
    - Dépendance avec d'autres logiciels ;
    - Ressources nécessaires et dédiées pour l'intergiciel ;
- Interface utilisateur : interface Web et/ou programme spécifique ;
    - Soumission de travaux ;

- Accès aux résultats des travaux ;
- Accès aux données ;
- Accès aux ressources ;
- Accès à l'état du système ;

Services

Nous entendons par services, l'ensemble des prestations assurées par la société qui supporte l'intergiciel.

- Support
  - Centre d'appel dédié au support d'utilisation et de mise en œuvre de la solution ;
  - Outil de suivi des incidents : base de données permettant le suivi des incidents clients ;
  - Base de données statiques d'incidents et de leur résolution consultable par moteur de recherche ;
  - Documentation de mise en œuvre, d'utilisation et d'exploitation de l'outil ;
  - Langue de support : langue dans lequel le support peut être obtenu pour l'ensemble des critères précédents ;
- Expertise et conseil
  - Mise en œuvre : capacité des équipes à prendre en charge toutes ou une partie des phases de mise en œuvre de la solution (installation, support, maintenance) ;
  - Portage d'application : capacité des équipes de prendre en compte le portage de projets scientifiques, compréhension du besoin fonctionnel, mise en œuvre ;
  - Localisation des ressources : mobilisation et mobilités des équipes ;
  - Langue de travail : capacité de travail en français ;

Coûts

Dans cet axe d'évaluation apparaissent les coûts des prestations liés à l'utilisation de l'intergiciel. Nous envisageons toutes les possibilités.

- Licences
  - Coût d'acquisition de la licence d'utilisation pour une durée initiale de 3 ans ;
- Coût annexe de la mise en œuvre de la solution
  - Matériel nécessaire à l'installation de la solution ;
  - Logiciel et Système d'exploitation nécessaires ;
- Mise en œuvre

  - Installation ;
  - Paramétrage ;
  - Formation ;
- Portage d'une application scientifique
  - Prix journalier d'une ressource d'assistance au portage d'une application scientifique ;
- Exploitation
  - Administration et support : charges liées à l'administration et au maintien de la plate-forme ;

Risques

Les critères qui entrent en compte pour les risques sont essentiellement des critères comptables et économiques qu'une société doit se poser avant de conclure un marché avec une autre société. Cet axe revêt une importance certaine car il permet de juger sur des critères moins techniques. Les risques mettent en perspective des éléments qui contribueront au succès du programme Décrypthon.

- Société
  - santé, pérennité : stabilité financière de la société ;
  - effectifs de la société ;
  - capacité à mobiliser les ressources nécessaires pour le projet ;

Solution proposée
  - stabilité, fiabilité et maturité de la solution ;
  - flexibilité : capacité du produit à adresser les besoins spécifiques au projet ;
  - évolutivité : plan de développement produit, stratégie de l'éditeur ;
  - référence : autre plate-forme utilisant cette solution ;

Mise en œuvre
  - respect des délais : capacités à s'engager et à respecter les délais de mise en œuvre ;
  - aide et facilité de la mise en œuvre ;

Les solutions retenues et l'évaluation

De manière générale, les solutions retenues ont répondu à la problématique du programme Décrypthon avec une architecture identique, et avec un schéma d'implantation reposant sur les mêmes idées :

- un portail internet d'accès à la grille : travaux, données, *etc* ;
- une possibilité de soumission via un logiciel reposant sur un serveur central de grille ;

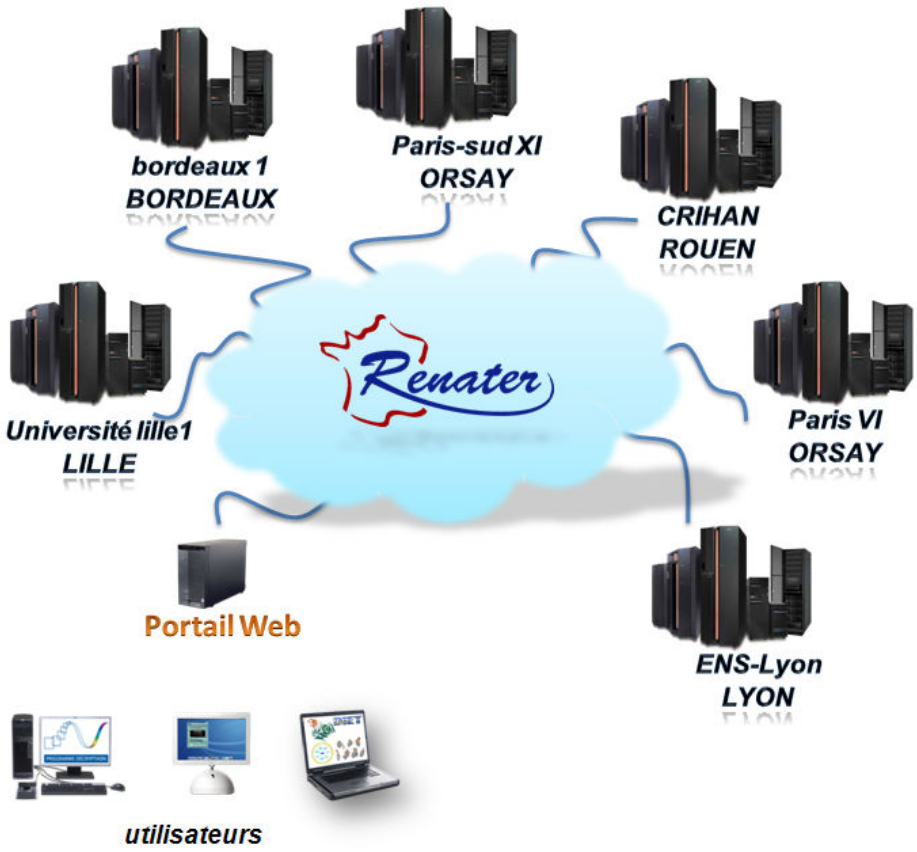


Figure 3.1 – Architecture générale de la grille Décrypthon.

– une implantation locale sur chaque centre de calcul universitaire d'outils logiciels permettant la réception et l'exécution des travaux de la grille Décrypthon ;

La figure 3.1 montre l'architecture générale de la grille Décrypthon, utilisant les 6 centres universitaires de calcul et un portail internet d'accès pour les utilisateurs des projets scientifiques.

Trois solutions commerciales ont été proposées par des sociétés éditrices d'intergiciel de grille, à savoir les sociétés *GridExpert*, *United Device* et *Platform computing*. Une quatrième solution a été envisagée, elle consistait à développer une solution spécifique pour le programme Décrypthon.

À chaque critère, nous attribuons une note qui s'échelonne de 0 à 3 :

– la note 0 signifie que l'intergiciel ne répond pas au critère ;
– la note 1 signifie que l'intergiciel ne répond pas ou partiellement à nos attentes, mais il existe un solution externe ;

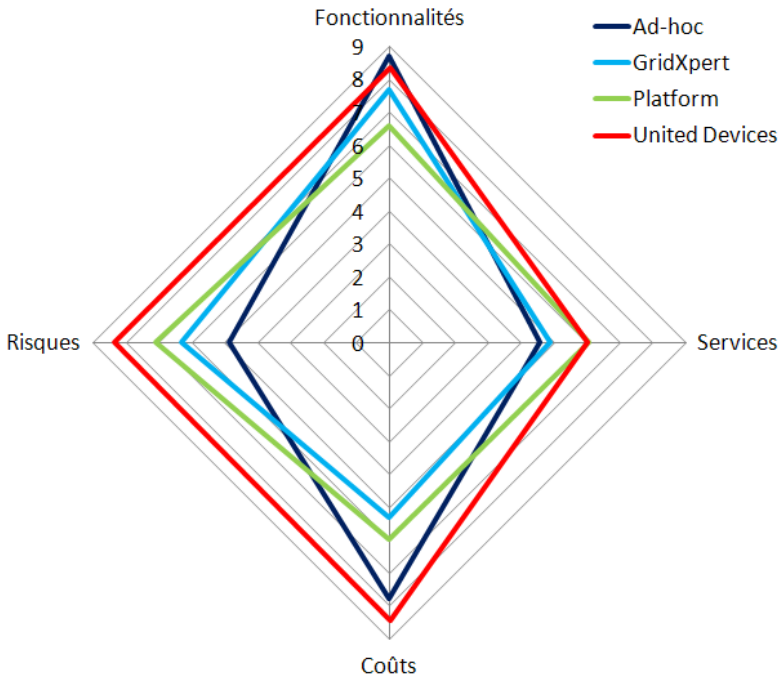


Figure 3.2 – Évaluation des solutions commerciales suivant 4 axes.

– la note 2 signifie que l'intergiciel répond au critère ;
– la note 3 signifie que l'intergiciel répond au delà des espérances au critère ;

Parallèlement à la notation, nous avons instauré une échelle de pondération qui s'étale de 1 à 5 : la valeur 1 accorde peu d'importance, la valeur 5 indique que le point évalué est déterminant. Pour chaque critère, la note est pondérée par un coefficient. Le total des notes pondérées obtenu pour les critères d'un thème est ensuite à nouveau pondéré par un coefficient. Au final nous obtenons une note ramenée à 10 qui donne la note obtenue par chaque solution évaluée suivant les 4 axes d'analyse (voir la figure 3.2).

Cette évaluation a été menée en début d'année 2005. Il ne nous semble pas nécessaire ni utile de dévoiler les pondérations que nous avons utilisées. En effet la pondération accordée à un critère et au thème auquel il appartient constitue une part subjective de l'étude. L'échelle d'importance octroyée aux différents points traduit une vision de la grille suivant les besoins exprimés par le programme Décrypthon. Cette étude est à placer dans le contexte du programme Décrypthon et n'a de sens que dans celui-ci.

Les résultats de l'évaluation entre les 3 solutions commerciales et l'hypothétique solution *ad-hoc* sont présentés dans la figure 3.2.

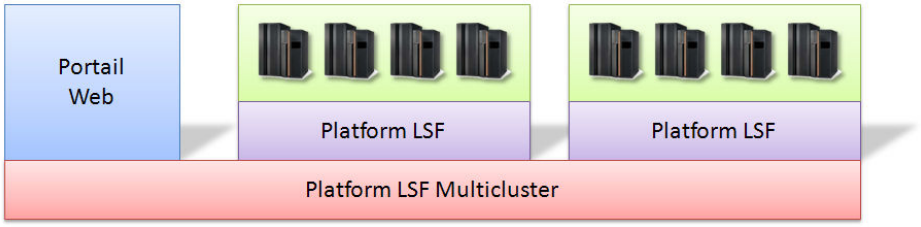


Figure 3.3 – Architecture prévisionnelle *Platform LSF / LSF MultiClusters*.

Les trois solutions commerciales proposées par *Platform computing*, *GridXpert* et *United Device* sont détaillées succinctement dans ce qui suit.

Platform computing

Platform LSF et *Platform LSF MultiClusters* sont les solutions commerciales proposées par la société *Platform computing* pour répondre à notre problématique. Chaque site comporte un ordonnanceur local (i.e *Platform LSF*), et l'extension *Platform LSF MultiClusters* permet la migration des travaux d'un site à l'autre. L'interface de soumission des travaux des utilisateurs se fait via une ligne de commande et/ou par l'accès à un portail internet. La figure 3.3 montre le schéma prévisionnel d'installation de la solution.

Cette solution répondait totalement à nos attentes, cependant, elle nous contraignait à installer l'ordonnanceur local *Platform LSF* sur chaque site entrant dans la grille universitaire. Cette nécessité était incompatible avec les logiciels déjà implantés localement sur les sites. Cette solution était trop intrusive au niveau des centres de calcul des universités.

GridExpert

GridXpert est une « start-up » créée en mai 2002, depuis août 2005, cette société a été rachetée par la société *United Device*. Sa solution *GX Synergy* se base sur l'ensemble des briques logicielles fourni par le *Globus Toolkit* 2.4 et des développements spécifiques faits par l'entreprise. La solution consiste en un *GridServer* et en des *GridAgents* répartis sur les sites de calcul. La gestion de la grille se fait par un portail Web après installation des composants nécessaires. Les *GridAgents* s'installent sur une machine frontale de chaque centre de calcul universitaire et soumettent les travaux issus du GridServer sur les ressources disponibles localement.

La soumission de nouveaux travaux peut se faire par l'intermédiaire d'une interface Web spécifique à développer ou d'un logiciel propriétaire à installer sur les machines des scientifiques. Cette solution était adaptée à nos besoins

et aux fonctionnalités désirées, cependant la société n'était pas compétitive en terme de services et de coûts associés à leur prestation. Cette solution n'a donc pas été adoptée.

United Device

United Device est une société fondée en 1999 au Texas, elle a développé son offre commerciale sur son produit phare GridMP basé sur les technologies issues des projets *distributed.net* et *SETI@home*. Elle destine son offre commerciale aux entreprises désirant utiliser leurs ressources sous-utilisées ou dormantes. Nous exposerons les détails de cette solution par la suite (section 3.3).

Cette solution a été choisie pour plusieurs raisons : les fonctionnalités offertes répondaient de manière suffisante aux besoins exprimés au moment de la conception de la grille universitaire Décrypthon. De plus, la solution proposait une implantation non invasive dans les centres de calcul. En effet elle ne demandait aucune configuration particulière de la part des administrateurs et aucun changement pour les utilisateurs locaux. En outre, les délais de mise en place et les services associés nous permettaient de respecter les délais de mise en production de la plate-forme Décrypthon. Enfin, la volonté affichée de la société d'être présente sur le marché européen et français a permis d'établir un partenariat entre le programme Décrypthon et *United Device*. La grille Décrypthon serait une vitrine et un faire valoir. Depuis septembre 2007, cette société s'appelle désormais *Univa UD* à la suite d'une fusion avec *Univa*.

La décision finale a été prise par le comité directeur Décrypthon à la lumière cette étude. Les trois solutions pouvaient chacune prétendre à gérer la grille Décrypthon. En dépit de l'adéquation parfaite qu'aurait connue une solution *ad-hoc* développée spécifiquement pour le programme Décrypthon, cette éventualité a été écartée en raison des délais de développement et des risques associés aux retards. La décision traduisait aussi la volonté affichée d'aider les projets scientifiques sélectionnés par le programme Décrypthon et non de développer un $n^{\text{iéme}}$ intergiciel de grille.

Le choix de la société *United Device* s'est donc avéré l'option la plus appropriée à nos besoins, bien qu'en terme de fonctionnalités, nous étions en dessous des offres proposées par les concurrents. Il s'est avéré que les services, le risque et le coût de leur solution étaient plus intéressants (voir la figure 3.2).

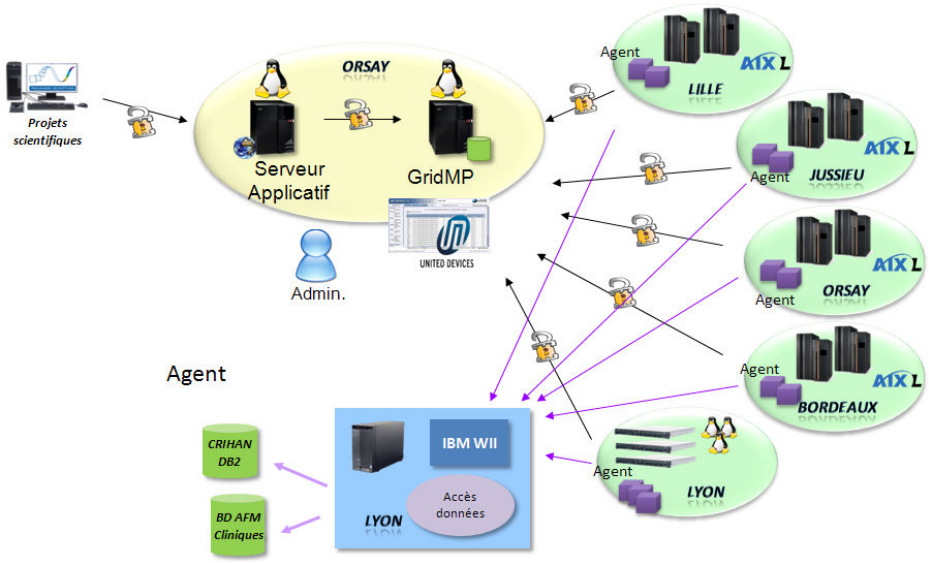


Figure 3.4 – Architecture de la grille Décrypthon version 1 (GridMP).

3.3 La grille Décrypthon version I

La première version de la grille Décrypthon a été réalisée à partir de la solution *GridMP* proposée par la société *United Device*. L'architecture de la solution telle qu'elle a été mise en œuvre est présentée par la figure 3.4. Elle s'articule autour :

- d'un serveur de grille *GridMP* qui centralise toutes les informations (les travaux à effectuer, les programmes exécutables et les données)
- d'un ensemble d'agents installés sur chaque site, chargés de venir chercher le travail auprès du serveur de grille. Ce mode de fonctionnement est appelé *pull model*.

Le serveur central de la grille (*GridMP*) définit 5 services principaux :

Realm Service : c'est le service d'authentification, toutes les opérations sur le serveur passent d'abord par l'obtention d'une autorisation (certificat) délivrée par ce service. Ainsi chaque communication est cryptée et les accès sont authentifiés.

Poll Service : ce service contient les instructions à donner aux agents lorsqu'ils se connectent au serveur de grille. Ces instructions peuvent être : nettoyer le cache de données, reconfigurer l'agent, obtenir des informations sur la machine, *etc.*

Dispatch Service : ce service distribue le travail aux agents lorsqu'ils se

connectent. Le travail attribué dépend des caractéristiques de la machine sur laquelle l'agent s'exécute et des contraintes exprimées par le travail à réaliser. Ce service est aussi responsable du maintien du statut des travaux effectués par les agents et du stockage des résultats dans le *file service*.

File service : ce service stocke le catalogue des données. Les données sont définies dans cette base de données. Elles peuvent être stockées directement par ce service, ou être décrites par une *URL* permettant d'y accéder.

Web service - MGSI [1] : il est disponible sur le serveur de grille. Il permet une abstraction de tous les autres services de gridMP afin de commander la grille. C'est par lui que tout passe : création de nouveaux travaux, envoi d'une donnée, récupération de résultats, suppression de travaux, *etc.*

3.3.1 Portage ou « gridification » d'une application

Le portage d'une application s'effectue en 2 étapes. Il faut d'abord construire une archive contenant l'environnement nécessaire à l'exécution de l'application (librairie, binaires, scripts, fichiers de configuration, *etc*). Cette archive est appelée *program module*. L'application est enregistrée au niveau du serveur de grille. La figure 3.5 schématise les différents niveaux de définition d'une application. Lors de l'enregistrement, il faut fournir un nom d'application, un nom de programme et sa version. Le *program module* est une instance concrète d'un programme. Une application peut comporter plusieurs programmes avec différentes versions. Il en est de même pour une version de programme qui peut avoir plusieurs *programs modules* déclinés en plusieurs versions.

À chaque enregistrement d'un *program module*, il est spécifié pour quel système d'exploitation l'archive a été construite. Ainsi, une application peut être exécutée sur n'importe quel système d'exploitation à partir du moment où il a été créé un *program module* et qu'il existe un agent pour ce système. De plus, il existe un ensemble d'outils permettant de vérifier le bon fonctionnement d'un *program module* sur les machines cibles. Ainsi un agent de test est utilisé pour exécuter localement le *program module* avec ces paramètres d'entrée avant la mise en production.

[1] Meta Processor Grid Services Interface

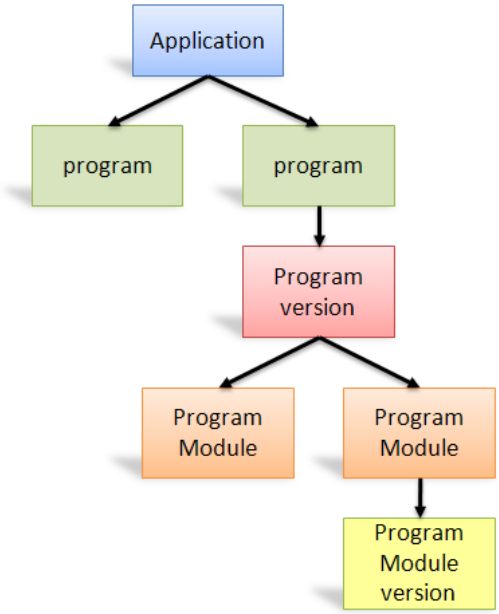


Figure 3.5 – Modèle de définition d'une application grille.

3.3.2 Gestion des données

Il existe un service spécifique de gestion des données dans le serveur *GridMP*. Ce service est un catalogue, entendez par là une base de données relationnelles, gérant l'identifiant et la description de la donnée. Une donnée est nécessairement un fichier archive contenant potentiellement plusieurs fichiers et un fichier XML qui décrit son contenu, on parle de *data package*.

Une donnée s'instancie de la manière suivante : Un *Data Set* regroupe un ensemble de *Data*. On retrouve les schémas de gestion des données dans la figure 3.6.

Toutes les commandes de gestion de données nécessaires existent au travers d'une interface de programmation (API) définie dans le service Web *MGSI*. Une gestion des droits est faite au niveau de chaque *Data Set*. Ainsi, une donnée peut être partagée entre tous les utilisateurs ou être seulement accessible à son propriétaire. De plus, le *Data Set* peut être associé ou non à un *job*. Dans ce cas, la visibilité et les privilèges des données seront hérités de celui-ci et lorsque le *job* est supprimé, les données associées le sont aussi. Enfin, pour chaque donnée est défini un niveau de cache [0-99], fixant la priorité de suppression dans le cas où il manquerait de l'espace disque au niveau des agents.

Nous décrirons par la suite le mécanisme de création d'un *job* qui mixe

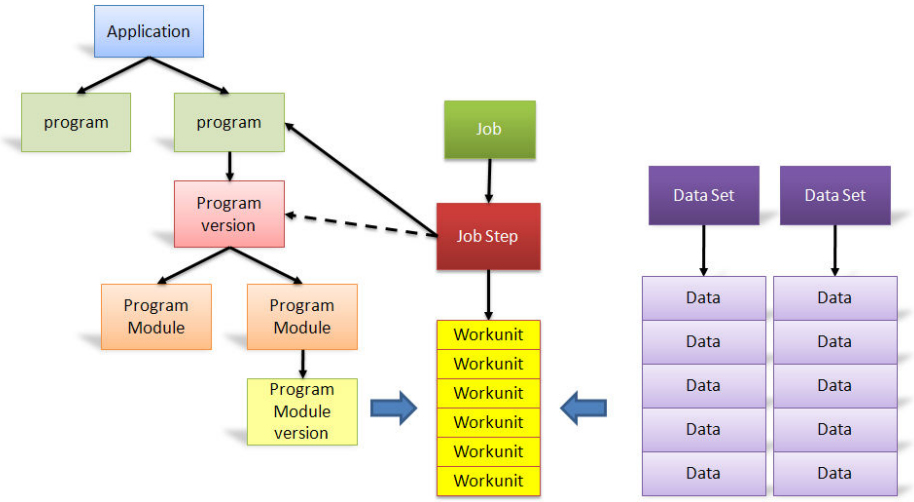


Figure 3.6 – Modèle de définition d'une application grille.

les données et les applications.

3.3.3 Création d'un *Job*

Il existe une interface de programmation (API) spécifique au niveau du serveur de grille qui permet de définir un *job* sur la grille *GridMP*. Ce *job* est composé de *jobStep* qui eux-mêmes se composent de *workunits*. Conceptuellement une *workunit* est l'association entre une ou plusieurs données (*data*) et un *program module* (voir la figure 3.6).

Les étapes de création d'un job pour une application préalablement enregistrée dans le serveur de grille s'effectuent de la manière suivante :

- Création des données paramètres du *job* : *data package* ;
- Création du *job* et du *jobstep* ;
- Création d'un (ou plusieurs) *data set* ;
- Création des *datas* et enregistrement des *data package* dans le *file service* ;
- Création des *workunits* dans un *jobstep*.

Toutes ces étapes se font via l'appel à l'API définie par le *MGSI*. À l'aide de l'API, nous avons alors développé un portail Web spécifique à chaque projet scientifique. Ce portail Web contient la logique de l'application scientifique et la logique de création (GridMP) du *job* associé. Si bien que les utilisateurs n'ont plus qu'à fournir les paramètres spécifiques de leur application et la soumission est effectuée automatiquement.

Le portail Web fournit une abstraction de la grille qui permet aux utilisateurs de ne pas se soucier des processus engendrés par la soumission d'un *job*. De plus, si l'application exploite un parallélisme de données, la logique de cette parallélisation est contenue dans le portail Web.

3.3.4 Fonctionnement de la grille

Le fonctionnement de la grille est assez simple. À chaque création d'un *job* est associée un *jobstep* et un ensemble de *workunits*.

Ces *workunits* sont prêtes à être lancées sur les ressources de la grille, elles contiennent les informations sur les données, les paramètres nécessaires ainsi que le programme à exécuter.

Les agents installés sur chaque machine de la grille se connectent à intervalle de temps régulier au serveur de grille pour prendre du travail (principe du modèle *pull*). Si les caractéristiques d'une *workunit* correspondent, alors les données associées à la *workunit* et le *program module* correspondant au système d'exploitation de l'agent sont téléchargés. Avant de télécharger les données, l'agent vérifie si elles ne sont pas déjà dans son cache évitant ainsi des transferts inutiles. L'agent lance alors le programme scientifique. À la terminaison du programme, l'agent archive les résultats (les résultats sont spécifiés par les paramètres de la *workunit*) et renvoie l'archive du résultat au serveur de grille.

À chaque *job* terminé est donc associé un ou plusieurs résultats (suivant le nombre de *workunits*). L'utilisateur télécharge alors l'ensemble des résultats par l'intermédiaire du portail internet, qui se charge, le cas échéant, d'agréger l'ensemble des archives résultats des *workunits* en une seule archive.

3.3.5 Transition

En début d'année 2007, la question du renouvellement en fin d'année des licences du logiciel *United Device* s'est posée. Nous avions quelques fonctionnalités supplémentaires que nous voulions ajouter à la grille. En effet le fonctionnement de la grille Décrypthon dans sa première version n'utilisait pas les ordonnanceurs locaux (*batch scheduler*) des centres de calcul. Au lieu de ça, l'agent était lancé en interactif sur les ressources Décrypthon. Nous utilisions donc bien les ressources propres au Décrypthon, les utilisateurs locaux pouvaient aussi utiliser les ressources Décrypthon au risque d'avoir leurs travaux tués ou ralentis si un *job* Décrypthon arrivait. Plusieurs alternatives s'offraient à nous :

– Apporter des changements à l'agent de sorte qu'il puisse soumettre les *workunits* dans les ordonnanceurs locaux.

– Lancer des agents dans les queues des ordonnanceurs locaux de manière chronique afin d'utiliser de temps en temps les ressources des universités.

Parmi les deux améliorations précédentes, la première n'était pas envisageable, car le code source de l'agent n'était pas ouvert, nous ne pouvions donc pas le modifier. La deuxième solution n'était pas acceptable car elle pouvait dans une période de faible activité des projets scientifiques gaspiller les ressources des universités.

De son côté, la société *United Device* nous proposait de migrer notre intergiciel grille vers leur solution Synergie nouvellement acquise (ex solution de *GridXpert*) plus adaptée aux orientations que la grille Décrypthon avait suivies. Ce changement impliquait une refonte totale du portail internet et de nouveaux coûts de migration.

Nous avons alors posé la question différemment :

> *Ne pouvons-nous pas nous tourner vers une solution* libre *durant l'année 2007 afin de remplacer progressivement l'intergiciel propriétaire de grille ?*

De plus, l'adoption d'une solution grille *libre* pouvait permettre d'apporter des modifications spécifiques aux besoins du projet, tout en profitant à tout un chacun.

C'est pourquoi nous avons étudié une nouvelle fois les solutions grilles possibles, en nous limitant aux solutions *libres*. Cette fois, le contour des besoins était bien plus précis et nous avions presque un an devant nous pour mettre en place une solution. Deux choix d'intergiciel ont retenu notre attention :

– l'intergiciel de grille gLite utilisé et développé dans le projet européen EGEE.
– l'intergiciel DIET développé dans l'équipe GRAAL (LIP, ENS-Lyon).

Il est évident que bien d'autres solutions étaient disponibles. Nous nous sommes tournés vers les solutions que nous connaissions le mieux.

Techniquement l'intergiciel gLite n'est pas très éloigné de la solution commerciale *Synergy* de la société *Univa UD*. Les deux intergiciels se basent sur le *Globus Toolkit* et ils ont développé certaines fonctionnalités qui leur sont propres.

La figure 3.7 montre un schéma d'implantation de l'intergiciel gLite dans le contexte de la grille Décrypthon. Apparaissent sur cette figure les différents composants gLite que nous aurions utilisés pour mettre en place la grille Décrypthon. Même si la volonté de l'intergiciel est de devenir une solution *libre* dans le but de rendre possible la mise en place d'une grille autonome, cet intergiciel est encore loin d'être à ce stade de maturité. Ainsi pour pouvoir utiliser les composants gLite, il est préférable de créer une organisation vir-

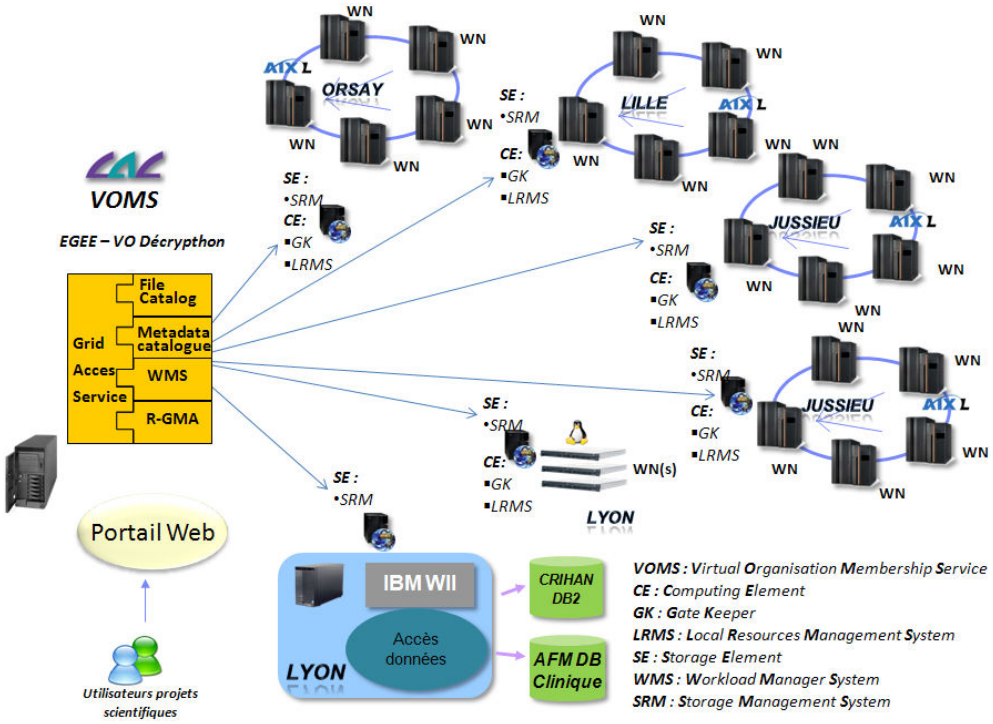


Figure 3.7 – Architecture gLite pour la grille Décrypthon.

tuelle (VO) sous l'égide du projet EGEE.

À partir de cette hypothèse nous avons imaginé quelle aurait été l'implantation de gLite sur les ressources existantes. Ainsi sur chaque site, nous aurions installé une machine supplémentaire avec les différents services de gLite, à savoir :

- un *Computing Element* (CE) avec un *Gate Keeper* (GK) qui permet de recevoir les travaux soumis par les utilisateurs, et son interface de soumission au gestionnaire local de ressources (LRMS) ;
- un *Storage Element* avec son interface de gestion des données (SRM).

Enfin sur les machines qui servent de point d'entrée de la grille nous aurions installé :

- le service de gestion d'ordonnancement des travaux : Workload Manager System (WMS) ;
- le service d'information sur les ressources du système : Relational Grid Monitoring Architecture (R-GMA) ;
- le service de gestion des données : LCG[2] File Catalog (LFC).

[2]LHC Computing Grid (LHC = Large Hadron Collider)

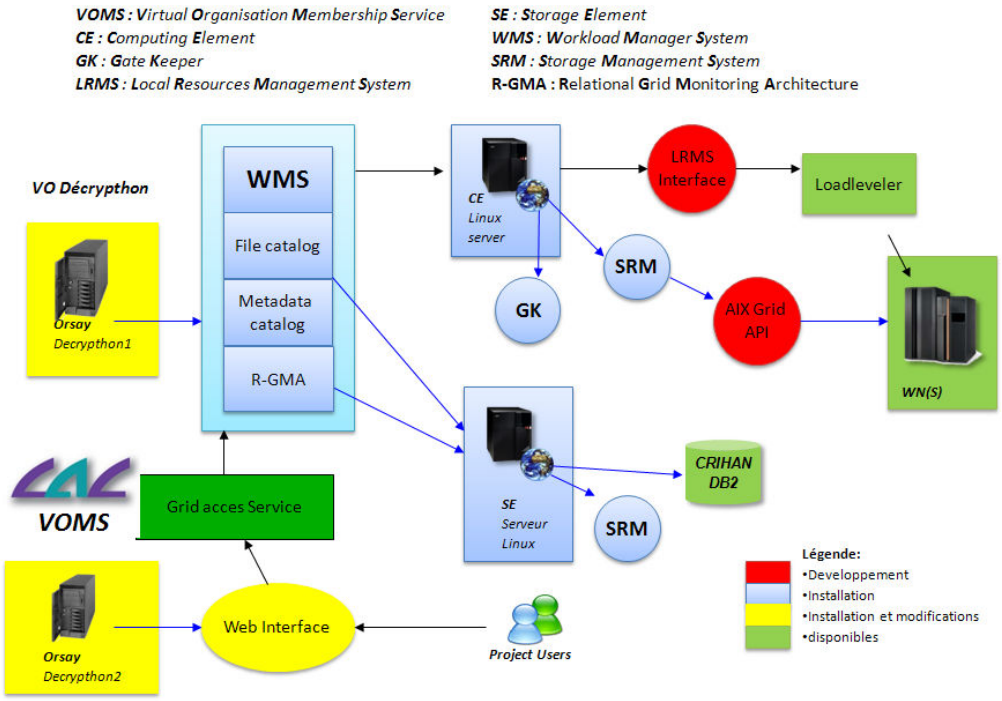


Figure 3.8 – Mise en place gLite pour la grille Décrypthon.

Outre l'orchestration générale de l'implantation de gLite présentée dans la figure 3.7, le choix de l'intergiciel gLite aurait demandé un certain nombre de développements pour être en mesure de rendre la grille opérationnelle. La figure 3.8 montre les développements nécessaires. Ces développements auraient été effectués au niveau de l'interface de soumission d'un *job* sur les gestionnaires locaux de ressources *Loadleveler*, et au niveau des bibliothèques de gestion des données pour le système AIX. Et bien sûr il aurait été nécessaire d'adapter le portail Internet pour permettre aux utilisateurs de la grille de soumettre leurs travaux.

D'autre part, la mise en production de gLite au sein de la grille Décrypthon aurait demandé l'installation de nouvelles machines dans la configuration actuelle des centres de calcul. En effet les bibliothèques disponibles pour l'installation des différents composants gLite sont disponibles uniquement pour le système d'exploitation *Scientific Linux 3*, or toutes les machines du Décrypthon sont exploitées par un système AIX.

De plus, même s'il existe une volonté et des moyens mis en place pour la formation à l'utilisation et à l'installation de gLite (GILDA), il ne semble pas clairement établi que le support puisse être fourni dans un contexte différent

des sites conventionnels EGEE.

Enfin, il est actuellement très difficile d'utiliser gLite comme intergiciel sans pour autant faire partie du projet EGEE, c'est pourquoi nous avons directement pensé à une implantation avec une organisation virtuelle (VO) Décrypthon s'appuyant sur l'infrastructure déjà définie au LAL[3]. Or, le programme Décrypthon désire garder son autonomie, même si nous trouverions un intérêt certain à entrer dans le projet EGEE en tant qu'organisation virtuelle (VO).

Tous ces arguments nous ont donc poussé à envisager une autre solution. Cependant, il n'est pas exclu qu'un jour le programme Décrypthon vienne incorporer le projet EGEE en tant que VO, ou encore incorpore une VO existante (par exemple la VO Biomed). Mais cette décision relèvera d'une volonté politique du comité directeur du programme Décrypthon.

Au cours de l'année 2007, nous avons décidé de migrer progressivement la grille Décrypthon en nous basant sur l'intergiciel DIET. Nous le verrons dans la description de la grille Décrypthon version II (c.f. section 3.5). DIET est un intergiciel modulaire, léger et complètement paramétrable qui permet de s'adapter facilement dans un contexte hétérogène. De plus, l'engagement de l'équipe de développement de l'intergiciel dans le support aux besoins de la grille Décrypthon a été un facteur déterminant dans cette décision. Enfin DIET ne nécessitait aucune modification, matérielle ou logicielle dans l'environnement de la grille Décrypthon. Nous avons même pu faire coexister les deux intergiciels pendant la phase de transition, ce qui a rendu le processus de migration transparent pour les utilisateurs.

En juin 2007, la décision a été prise par le comité Directeur Décrypthon de mettre en place le prototype de la grille Décrypthon avec l'intergiciel DIET. Dans la suite de ce chapitre nous décrivons l'intergiciel DIET dans son ensemble (section 3.4), puis nous montrerons comment nous avons bâti la grille Décrypthon version II (section 3.5) grâce à DIET, qui est devenu en décembre 2007 l'intergiciel officiel de production de la grille universitaire Décrypthon.

3.4 DIET et son écosystème

Plusieurs approches existent pour développer et mettre en place des applications dans une grille informatique : passage de messages (MPI), gestionnaire de queue, portail Internet, et les *Network Enabled Server* (*NES*). En substance les *NES* se décrivent de la manière suivante : des clients soumettent des requêtes de calcul à un ordonnanceur qui est chargé de localiser un ou

[3]Laboratoire de l'accélérateur Lunéaire à Orsay

plusieurs serveurs offrant ce service sur la grille. L'ordonnanceur nommé parfois agent est utilisé pour équilibrer la charge et retourner aux clients la liste des serveurs disponibles. Les clients envoient alors les données nécessaires à l'exécution de leur service en choisissant un serveur parmi la liste suggérée.

Le projet *Distributed Interactive Engineering Toolbox* (DIET) [93] s'est concentré sur le développement d'un intergiciel de grille de type *NES* respectant le standard *GridRPC*. Ce standard est l'implantation dans un environnement grille du paradigme classique d'appel de procédure *Remote Procedure Call*. Le gridRPC [85] est défini par un travail commun au sein de l'*Open Grid Forum* (OGF). DIET est constitué de plusieurs composants logiciels qui permettent de construire une application répartie sur une grille. L'intergiciel base sa recherche sur la description de la requête du client (nom du service, paramètres et taille des données), les capacités de traitement des serveurs et la localité des données utilisées pour la résolution du service. Un mécanisme de gestion de données est pourvu, permettant de stocker de manière permanente ou temporaire les données dans l'intergiciel en vue d'une utilisation future pour une autre requête. D'autres *NES* ont été développés. Parmi ces *NES* qui implantent les standards du GridRPC nous retrouvons DIET [4], NetSolve [10], Ninf [70] et omniRPC [83]. L'originalité de DIET repose entre autre sur une architecture hiérarchique d'agents.

3.4.1 L'architecture de DIET

DIET est basé sur un modèle Client-Agent-Serveur. Il se différencie notamment par la distribution d'agents en une hiérarchie d'agents.

On distingue parmi les agents, le *MasterAgent* (MA) et les *LocalAgent* (LA). Cet arbre d'agents est enraciné sur le MA qui est le point d'entrée pour un client. Chaque agent (MA) et (LAs) peut être relié à des *Server-Deamons* (SeDs) qui sont les points d'entrée aux ressources de calcul et de stockage. Plusieurs arbres d'agents peuvent être interconnectés dynamiquement ou statiquement à d'autres par l'intermédiaire de chaque point d'entrée (i.e les MA) de chaque hiérarchie. La figure 3.9 montre une architecture complète formée de plusieurs hiérarchies d'agents. Le client contacte son point d'entrée grâce au service de nommage standard CORBA, dans lequel chaque MA s'enregistre avec son nom.

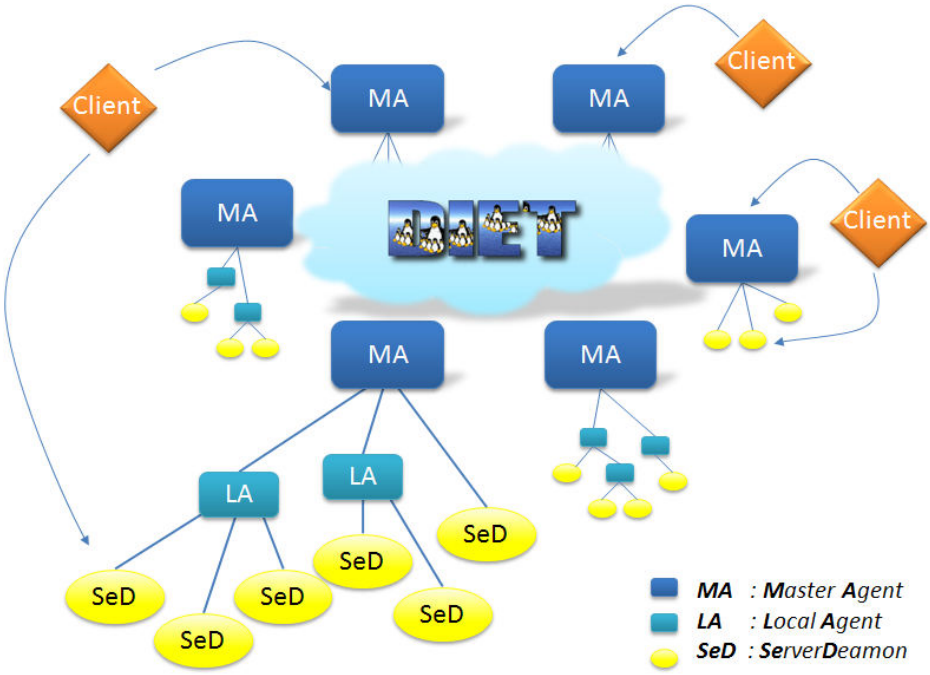


Figure 3.9 – Architecture DIET.

3.4.2 Fonctionnement de DIET

Portage ou « gridification » d'une application

DIET fournit un ensemble d'interfaces permettant à des développeurs de publier leurs applications sur la grille, et aux clients (les utilisateurs) de faire appel à celles-ci. DIET fournit une abstraction de l'application afin de la rendre disponible sur un ensemble de ressources réparties sur une grille. La figure 3.10 présente ces différents niveaux d'abstraction.

Typiquement, on veut rendre l'application disponible sur plusieurs ressources afin de pouvoir satisfaire plusieurs clients qui ne peuvent pas exécuter localement l'application. Avant toute chose, il faut disposer d'une application exécutable sur les ressources accessibles dans la grille, c'est l'étape (a) dans la figure 3.10.

Ensuite, « gridifier » une application dans l'intergiciel DIET comporte deux étapes :

– écrire la couche applicative serveur (étape (b) dans la figure 3.10). Cette étape consiste à coder à l'aide des fonctions offertes dans l'API DIET SeD [94] l'application serveur qui sera publiée dans la hiérarchie DIET. Il faut fournir l'ensemble des paramètres nécessaires au lancement de

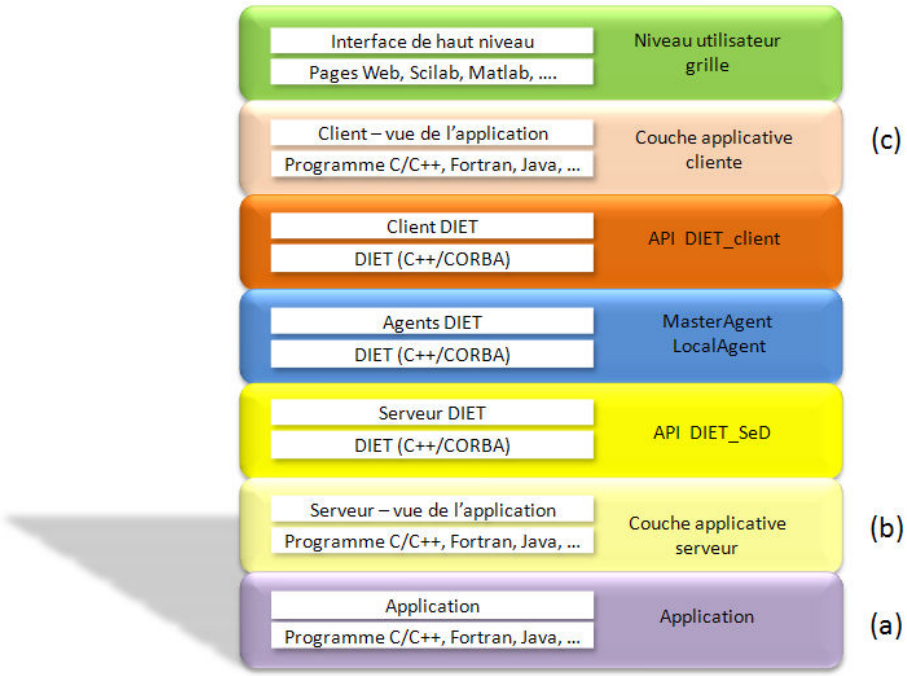


Figure 3.10 – Différents niveaux d'abstraction DIET.

l'application, et donner un nom à ce service. Cette couche applicative est l'interface intermédiaire entre le SeD DIET (qui fait partie de l'intergiciel) et le réel code applicatif. Le résultat de ce code après compilation, produit un binaire qui utilise les bibliothèques DIET SeD et donne accès aux fonctionnalités du cœur de DIET que nous décrirons dans les pages suivantes.

– écrire la couche applicative cliente (étape (c) de la figure 3.10) : cette étape consiste à coder à l'aide des fonctions de l'API DIET Cliente [94], l'appel au service (i.e l'application) déclaré dans la hiérarchie. Le développeur doit connaître les paramètres nécessaires pour l'appel de l'application distante. Il doit aussi connaître le nom du service associé.

Une fois les deux étapes précédentes effectuées, il peut être fourni une interface de plus haut niveau permettant l'accès à la grille directement depuis une page internet ou directement dans une application tierce telle que Scilab [43]. Cette interface est construite sur la couche applicative cliente. Nous exposerons dans les paragraphes qui suivent le détail du fonctionnement du cœur de DIET qui explique le paradigme du gridRPC [client_DIET - Agents (MA, LAs) - SeD], tel qu'il a été implanté dans l'intergiciel DIET.

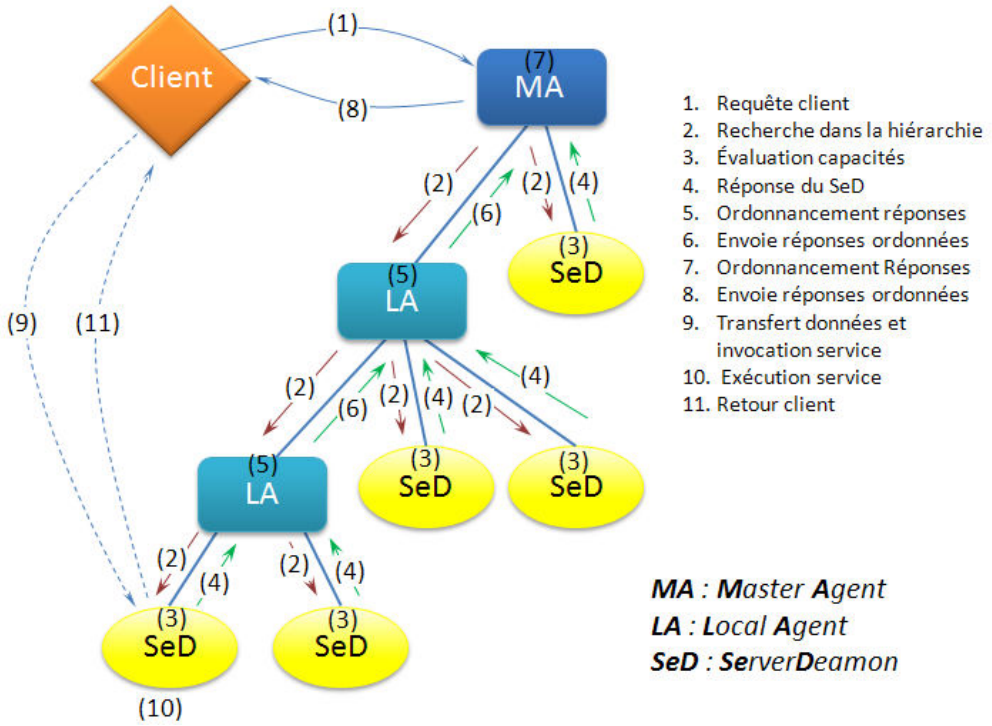


Figure 3.11 – Mécanisme d'une requête DIET (*diet_call*).

Soumission d'une requête DIET

La soumission d'une requête par un client DIET à un ensemble de ressources gérées par DIET est une succession d'étapes que nous illustrons dans la figure 3.11. Au niveau de la couche applicative cliente, toutes ces étapes sont masquées dans l'appel d'une unique fonction *diet_call*, c'est le fonctionnement du cœur de DIET basé sur le modèle NES et le standard gridRPC (voir figure 3.10).

La première étape (1) est la soumission du problème : le client se connecte au *Master Agent* (MA) et lui transmet une description de son problème. Le *Master Agent* parcourt alors la hiérarchie DIET en s'adressant à ses fils en leur transmettant la description du problème afin de trouver les SeDs capables de le résoudre (étape 2). Lorsque la requête arrive sur un SeD, il évalue sa capacité à traiter la requête (étape 3) et transmet à son père sa réponse contenant l'estimation de ses performances pour cette requête (étape 4). Lorsqu'un *Local Agent* (LA) a reçu toutes les réponses de ses fils, il les ordonne en fonction des performances des SeDs (étape 5), et transmet sa réponse à son père (étape 6). Lorsque le *Master Agent* a reçu les réponses

mode	description
DIET_VOLATILE	non persistant
DIET_PERSISTENT	persistant et déplaçable
DIET_PERSISTENT_RETURN	persistant, déplaçable, copie est envoyée au client
DIET_STYCKY	persistant, non déplaçable sur un autre serveur
DIET_STYCKY_RETURN	persistant, non déplaçable, copie est envoyée au client

Tableau 3.1 – Mode de persistance de données dans DIET.

de l'ensemble de ses fils, il ordonne à son tour les SeDs capables de résoudre le problème (étape 7) et envoie cette liste ordonnée au client (étape 8). Le client contacte alors le SeD, transfère ses données si nécessaire, et invoque le service lié à son problème (étape 9). Une fois que le SeD choisi a récupéré tout ce dont il a besoin pour exécuter la requête, il la traite (étape 10). Les résultats sont renvoyés au client ou stockés dans la plate-forme (étape 11).

Gestion des données : DTM

Le service de gestion de données [29] *Data Tree Manager* (DTM) a été développé spécifiquement pour la plate-forme DIET. Ce système propose une gestion des données basée sur deux éléments clés : des identifiants pour les données et une gestion en arbre qui colle à la hiérarchie DIET. Dans le but d'éviter des transmissions inutiles de la même donnée d'un client vers les serveurs de calcul, DTM ajoute la possibilité de laisser les données à l'intérieur de la plate-forme après la fin d'une requête. Un identifiant est alors attribué à la donnée et retourné au client. Il pourra s'en servir pour faire référence à sa donnée lors d'une autre requête. Un client peut choisir, pour chacune de ses données, si elle doit être persistante ou non à l'intérieur de la plate-forme. Plusieurs modes de persistance sont proposés et sont décrits dans le tableau 3.1, la gestion des données est séparée de celle des requêtes. Le DTM est construit autour de deux entités, le *LocManager* et le *DataManager*.

- Un *LocManager* est situé sur chaque agent DIET avec lequel il communique directement. Ils sont organisés hiérarchiquement en arbre comme les agents DIET. Les *LocManagers* ont la charge de localiser les données dans la hiérarchie. Chaque *LocManager* stocke une liste de données présentes dans son sous-arbre.
- Les *DataManagers* sont situés sur chaque *SeD*. Ils stockent les don-

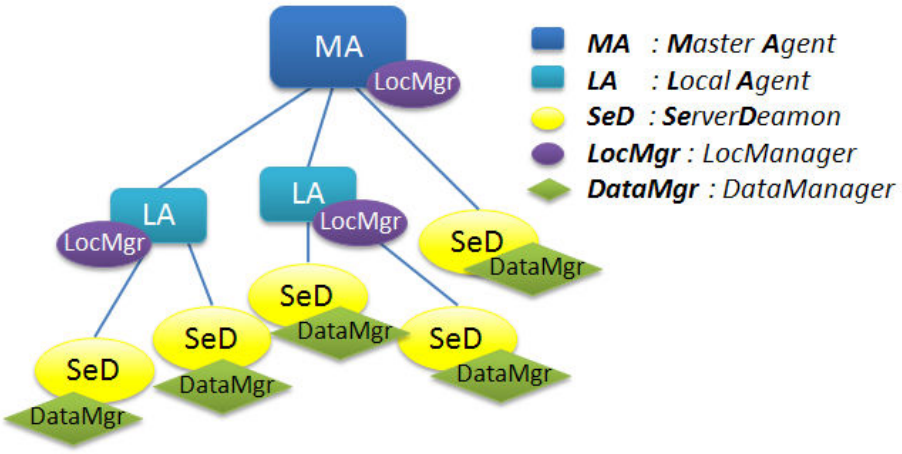


Figure 3.12 – Gestion des données dans DIET (*Data Tree Manager* : DTM).

nées persistantes et maintiennent une liste d'identifiants de celles-ci. Lorsqu'un SeD a besoin d'une donnée persistante, il la demande à son *DataManager* qui la cherche dans la hiérarchie des *LocManagers* et la lui fournit. Enfin, un *DataManager* informe son *LocManager*, situé au dessus, de toutes les opérations de transfert effectuées sur les données persistantes.

La figure 3.12 montre l'organisation du DTM au sein d'une hiérarchie DIET.

Cette gestion des données est intégrée dans le cœur de DIET, l'utilisation de l'API DIET rend les transferts transparents du point de vue développeur applicatif client et serveur. La gestion des données au sein de DIET est ouverte. Ainsi, il a été développé et ajouté des interfaces permettant à DIET d'utiliser JUXMEM [9], un gestionnaire de données pair-à-pair indépendant offrant un partage cohérent, un stockage persistant et une tolérance à la volatilité. Actuellement, il est aussi possible de déléguer la gestion des données persistantes à DAGDA [94] qui donne un plus grand contrôle quant aux politiques de placement, de réplication, *etc.*

Ordonnanceur spécifique : Plug-in Scheduler

Par défaut, l'ordonnancement établi dans DIET est le suivant : le SeD choisi est celui qui a exécuté une requête client il y a le plus longtemps. Ce qui revient à faire un quasi *round-robin* sur toutes les ressources disponibles. Le mécanisme de sélection d'un SeD dans la hiérarchie DIET se base sur un vecteur d'estimation situé dans la réponse des SeDs vers les agents comme décrit dans le mécanisme de soumission d'une requête. Lorsque l'agent DIET

reçoit le vecteur d'estimation, il ordonne les réponses en fonction des valeurs contenues dans l'estimation. Ainsi, par défaut, l'ordonnanceur ordonne les réponses en classant par ordre décroissant le champs TIMESINCELASTSOLVE de l'estimation, qui représente le temps écoulé depuis la dernière résolution de service par le SeD.

DIET offre la possibilité aux développeurs de la couche applicative serveur, d'affiner l'ordonnancement fait par la hiérarchie DIET en fonction des spécificités de son application. Pour se faire, le codeur définit son propre vecteur d'estimation et spécifie le comportement (i.e l'ordonnancement) que la hiérarchie doit avoir pour trier les réponses fournies par les SeDs. Ce mécanisme est appelé *plug-in scheduler* [23]. Il est mis en place au moment de l'enregistrement du service dans la hiérarchie où l'on déclare :

- la fonction d'estimation qui remplira les valeurs contenues dans le vecteur d'estimation ;
- l'ordre des valeurs à considérer dans le vecteur d'estimation et le tri associé.

Pour mieux comprendre ce mécanisme de *plug-in scheduler*, la figure 3.13 donne un exemple d'utilisation. La fonction d'estimation remplit les valeurs CPUSPEED et LOADAVG qui représentent respectivement la vitesse processeur et la charge moyenne du processeur[4]. Les deux priorités d'agrégation des réponses sont définies dans cet ordre : priority_max(CPUSPEED) et priority_min(LOADAVG). Ce qui donnera la règle suivante au niveau de chaque agent de la hiérarchie : les réponses des SeDs pour le service sont d'abord triées par CPUSPEED croissant. S'il y a égalité pour la valeur CPUSPEED, elles sont alors triées par LOADAVG décroissant.

Dans notre exemple, l'ordre établi par l'agent sera SeD_3, SeD_1, SeD_2.

Collecteur d'informations sur les ressources : CoRI

En addition du mécanisme de *plug-in scheduler* décrit précédemment, DIET offre un collecteur modulaire d'informations sur les ressources gérées par le SeD appelé *CoRI*. Cette fonctionnalité est complémentaire à la possibilité de définir un *plug-in scheduler*.

En effet, afin de permettre de spécifier les valeurs sur lesquelles l'ordonnancement est effectué, cet outil définit un cadre générique permettant de coder un collecteur d'informations. Les développeurs de l'intergiciel DIET peuvent implanter plusieurs collecteurs d'informations basés sur des méthodes différentes. Ils permettront ensuite d'avoir des valeurs à partir desquelles le développeur applicatif pourra renseigner le vecteur d'estimation et

[4]établie sur les 10 minutes précédentes par exemple

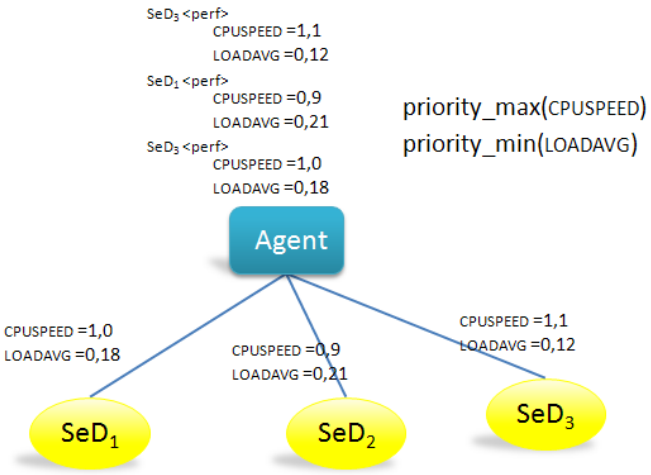


Figure 3.13 – Exemple de fonctionnement des ordonnanceurs spécifiques DIET.

définir son ordonnancement.

Ainsi, dans DIET on compte déjà le collecteur *CoRI-Easy* qui donne des informations basiques sur le nombre de *bogoMIPS*[5], la fréquence processeur, la mémoire disponible, *etc*, mais aussi le collecteur CoRi-FAST basé sur la prédiction de performance définie par FAST [78]. D'autres collecteurs d'informations seront implantés suivant les besoins liés aux politiques d'ordonnancement. Par exemple, les récentes prises de conscience sur l'énergie consommée par les machines pourrait faire émerger un collecteur donnant accès aux informations sur la consommation énergétique de la ressource.

DIET et les ordonnanceurs locaux

Les ressources de calcul de type grappe de machines peuvent être localement gérées par un système que l'on nomme *batch scheduler* comme par exemple Loadleveler [47], Torque [108], OAR [104], LSF [106], SGE [107], OpenPBS [105], *etc*.

Ce type de gestionnaires locaux de ressources accepte des travaux issus d'utilisateurs par l'intermédiaire d'une commande et d'un script décrivant l'exécution de l'application. Dans ce cas de figure, le rôle d'un SeD DIET est de fournir un traducteur commun à tous ces systèmes afin de rendre possible la soumission à ce type de ressources. DIET offre cette possibilité et permet de gérer l'exécution des applications au travers de ces gestionnaires locaux.

[5]le nombre de millions de fois par seconde qu'un processeur peut ne rien faire

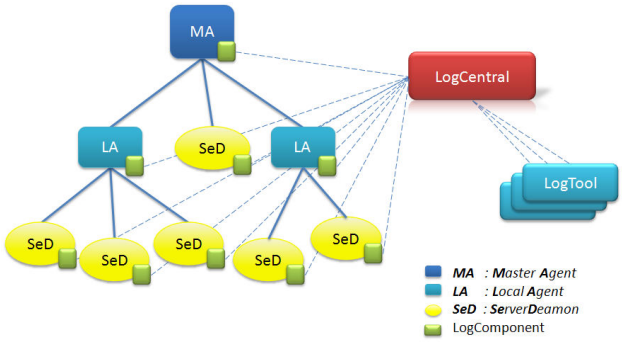


Figure 3.14 – Organisation hiérarchique de DIET avec les *LogComponents*.

Actuellement DIET supporte les ordonnanceurs locaux Loadleveler et OAR, et il est envisagé de soutenir d'autres types de gestionnaires (SGE et PBS).

3.4.3 Visualisation et surveillance

LogService [14] est un système de monitoring qui collecte et diffuse les messages de composants distribués. Il s'agit d'un système générique qui s'interface avec une application distribuée pour suivre son état et son activité. Ce service se compose de trois éléments. Les *LogComponents* qui sont les capteurs. Le *LogCentral* qui collecte les messages des *LogComponents*. Enfin les *LogTools* à qui sont distribués les messages collectés par le *LogCentral*.

À l'aide de ce service, les composants DIET (agents et SeD) ont été instrumentés avec des *DIETLogComponents* qui envoient l'activité de ceux-ci au *LogCentral*. Nous pouvons choisir d'activer ou non le *DIETLogComponent* au moment du démarrage d'un composant DIET en le spécifiant dans le fichier de configuration. La figure 3.14 montre l'intégration du LogService dans l'environnement DIET.

Deux outils de type *LogTools* sont fournis avec DIET :

- *DietLogTool* qui récupère et stocke l'ensemble des informations reçues par le *LogCentral* dans un fichier ;
- *VizDIET* qui fournit une représentation graphique de la hiérarchie et des statistiques sur l'activité sur la plate-forme.

VizDIET [14, 96] permet d'afficher l'ensemble des informations sur l'activité d'une plate-forme DIET, comme la charge des nœuds, le flux des requêtes ou encore le diagramme de Gantt de l'utilisation des services des SeDs et de la plate-forme. Cet outil de visualisation fournit un ensemble d'informations statistiques. Elles servent à l'analyse de la plate-forme et de ses éléments. Dans certains cas, VizDIET est aussi utilisé comme débogueur. En visua-

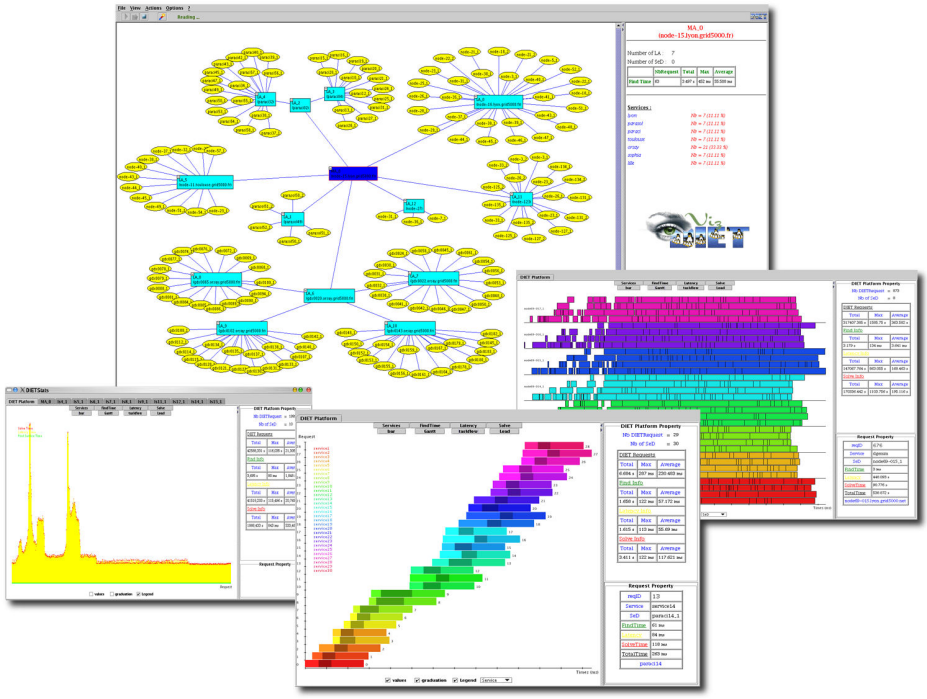


Figure 3.15 – Captures d'écrans du logiciel VizDIET.

lisant directement le diagramme de Gantt d'un ordonnancement ou le flux d'exécution des requêtes, nous pouvons vérifier d'un simple coup d'œil le bon fonctionnement d'une heuristique mise en place grâce aux ordonnanceurs spécifiques(*plug-in schedulers*). Un montage de plusieurs captures d'écran est donné dans la figure 3.15.

3.4.4 Déploiement d'une hiérarchie DIET

L'utilitaire *GoDIET* [24, 95] est un outil permettant le déploiement automatique d'une plate-forme DIET à l'aide d'un fichier XML décrivant les ressources, les services et la topologie de la plate-forme à déployer. *GoDIET* est chargé de générer et de copier les fichiers de configuration de chaque élément de la plate-forme DIET sur la ressource choisie. Une fois le transfert des fichiers effectué, *GoDIET* lance un à un les composants DIET (MA, LA, SeD).

Étant donné la structure en arbre de la hiérarchie DIET, *GoDIET* respecte l'ordre de traversée de l'arbre avant d'initialiser les éléments de la plate-forme. Il lance le père avant d'exécuter un fils. D'autre part, la hiérarchie

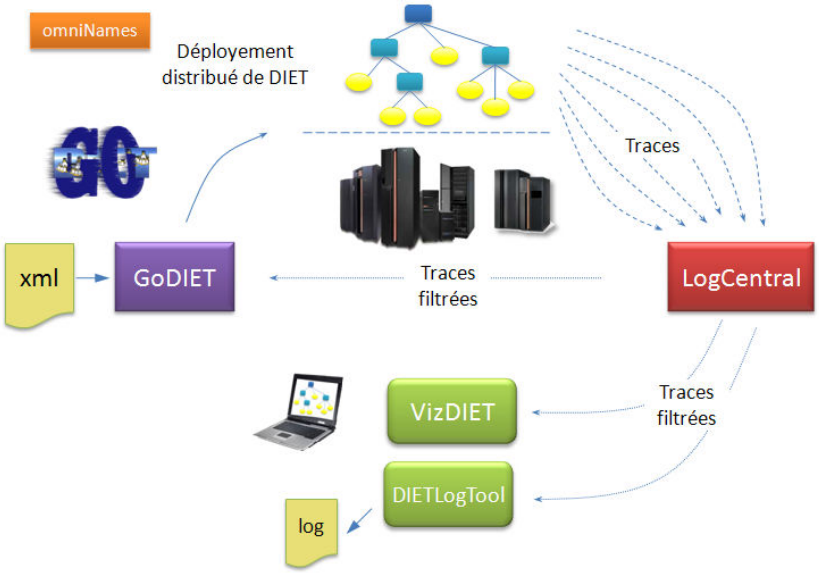


Figure 3.16 – Déploiement d'une hiérarchie DIET avec GoDIET.

DIET est dépendante du service de nommage CORBA (*omniNames*), ce service est donc initialisé en premier. De plus, si l'on veut activer la surveillance de la plate-forme (LogService), celui-ci est démarré avant tout composant DIET.

Dans le but de contrôler le déroulement du déploiement de la plate-forme, l'outil *GoDIET* peut se connecter au LogService en tant que *LogTool*. Il obtient alors un retour sur l'état de l'élément qu'il vient de déployer. La figure 3.16 montre le schéma de fonctionnement de *GoDIET*.

Le déploiement d'une hiérarchie DIET peut aussi être assuré par d'autres outils génériques de déploiement comme ADAGE [60].

3.5 La grille Décrypthon version II

Nous présentons comment nous avons utilisé l'ensemble des possibilités offertes par DIET dans le but d'implanter et de supporter la grille Décrypthon. À partir du mois de juillet 2007, nous avons progressivement mis en place l'architecture DIET sur les ressources. L'objectif qui a guidé la mise en place de la grille Décrypthon sous l'intergiciel DIET a été la continuité de service. En effet, pendant toute la phase de migration, la grille était active pour les projets scientifiques et aucune interruption n'a été à déplorer. Nous avons assuré la disponibilité des moyens de calcul installés dans les centres

universitaires.

3.5.1 Les ressources Décrypthon

Les ressources Décrypthon sont composées de machines parallèles multiprocesseurs (4 à 16 processeurs physiques) sous le système AIX et d'une grappe de machines mono-processeur sous le système Linux. Dans le détail, les machines sont décrites dans le tableau 3.2. Elles sont très hétérogènes, réparties et gérées par deux sortes d'ordonnanceurs locaux différents (OAR et Loadleveler). Au total, 58 machines hétérogènes pour 488 processeurs composent les ressources de la grille Décrypthon.

Cependant, tous les processeurs ne sont pas adressables en même temps par les utilisateurs Décrypthon. En effet, il existe des règles définies par chaque ordonnanceur local. Nous pouvons envoyer autant de travaux que nous le voulons, cependant seul un nombre borné de processeurs peut être utilisé simultanément. Si bien que la grille Décrypthon ne peut compter au maximum que sur 191 processeurs. Ce qui représente déjà 4O % des ressources totales des 6 sites qui composent la grille Décrypthon.

Il existe des règles de filtrage au niveau des pare-feux des sites universitaires, permettant la communication entre toutes les machines, et uniquement entre elles, sauf pour les machines ayant une vocation à être ouvertes (i.e serveur Web).

Parmi les sites impliqués dans la grille, certains ne sont pas des sites de calcul, ils sont destinés au stockage des données ou à fournir un portail d'accès. Ainsi le site du CRIHAN (Rouen) est un site de stockage et d'accès aux bases de données Décrypthon [71]. Il en est de même pour une machine à Lyon et deux machines à Orsay qui jouent le rôle de portail internet d'accès à la grille Décrypthon.

3.5.2 Architecture de la grille Décrypthon

En se basant sur les fonctionnalités offertes par l'intergiciel DIET, nous avons construit une architecture de la grille Décrypthon en déployant :

- un serveur de nom CORBA (*omniNames*) sur une des machines dédiées installées à Orsay ;
- un MasterAgent sur une des machines dédiées à Orsay ;
- un SeD sur les frontales de chaque centre de calcul universitaire qui gère la soumission des travaux Décrypthon aux ordonnanceurs locaux (OAR et Loadleveler).

La figure 3.17 schématise cette architecture.

Ressource	OS	processeur	Nb proc	Mémoire
7	Linux	Intel(R) Pentium(R) 4 CPU 2.40GHz x86_32	1	1 Go
11	Linux	Intel(R) Xeon(TM) CPU 2.40GHz x86_32	1	1 Go
6	Linux	Intel(R) Pentium(R) 4 CPU 3.00GHz x86_32	1	1.5 Go
2	Linux	Intel(R) Xeon(TM) 4 CPU 3.40GHz x86_32	2	4 Go
1	AIX	PowerPC POWER5 64 bits 2.0 GHz	4	16 Go
18	AIX	PowerPC POWER5 64 bits 1.5 GHz	16	32 Go
4	AIX	PowerPC POWER5 64 bits 1.9 GHz	16	16 Go
4	AIX	PowerPC POWER5 64 bits 1.9 GHz	8	16 Go
1	AIX	PowerPC POWER5 64 bits 1.9 GHz	16	32 Go
1	AIX	PowerPC POWER5 64 bits 1.9 GHz	16	64 Go
1	AIX	PowerPC POWER5 64 bits 1.5 GHz	16	64 Go
1	AIX	PowerPC POWER4 64 bits 1.1 GHz	16	16 Go
1	AIX	PowerPC POWER4 64 bits 1.7 GHz	8	16 Go

Tableau 3.2 – Les ressources de la grille Décrypthon.

3.5.3 Le DIET_Webboard : Portail Web DIET

Nous l'avons vu dans la figure 3.10 définissant les niveaux d'abstraction DIET, un client DIET est destiné à être intégré dans une interface utilisateur de haut niveau lui permettant d'accéder à la grille de manière transparente. Par ailleurs, nous ne l'avons pas mentionné dans la description de l'intergiciel DIET, mais ce dernier ne gère pas d'utilisateurs. Enfin il n'y a pas d'historique stocké dans la hiérarchie sur les requêtes qui ont été exécutées par cette dernière.

Nous avons donc développé un portail internet de gestion d'une plate-

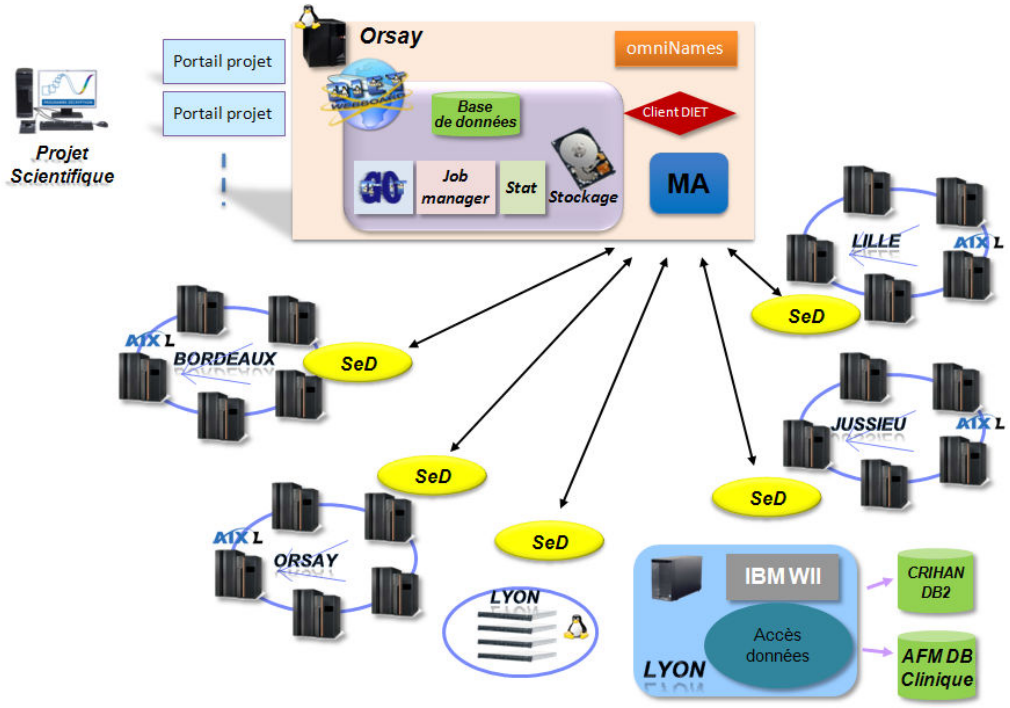


Figure 3.17 – Plate-forme Décrypthon.

forme DIET. Ce portail internet poursuit 5 objectifs principaux :

- conduire le déploiement de la hiérarchie DIET en se basant sur GoDIET ;
- gérer l'identification et les permissions d'un utilisateur accédant aux ressources Décrypthon ;
- permettre une abstraction des requêtes DIET en *jobs* et *workunits* dans le but d'exploiter le parallélisme de données.
- régir le déroulement de l'exécution des travaux utilisateurs en se basant sur l'intergiciel DIET ;
- exposer des statistiques d'utilisation des ressources et de la plate-forme Décrypthon.
- fournir une interface de programmation (API) pour écrire rapidement une interface de haut niveau pour l'utilisateur final afin qu'il puisse soumettre des travaux et récupérer ses données indépendamment de la plate-forme adressée.

Le DIET_Webboard est donc d'abord un portail internet d'administration d'une plate-forme DIET, intégrant les différents outils disponibles dans l'écosystème DIET pour fournir une abstraction de l'intergiciel et des ressources

de la grille. Il peut gérer plusieurs plates-formes à la fois : par exemple dans la grille Décrypthon, deux hiérarchies distinctes sont utilisées :

- la plate-forme dite de « production », elle est employée par les projets scientifiques ;
- la plate-forme de test, nous l'utilisons pour effectuer des tests sans perturber le bon déroulement de la plate-forme de production. Par exemple, lorsqu'une nouvelle application est intégrée dans la grille Décrypthon, elle est d'abord mise en test pour intégrer par la suite la plate-forme de production.

Les deux plates-formes sont complètement indépendantes, et peuvent néanmoins utiliser les mêmes ressources de calcul.

En outre, Le DIET_Webboard gère aussi les applications enregistrées et déployées dans la hiérarchie DIET. Il permet de déclarer les services (applications) disponibles au niveau de chaque SeD DIET et de vérifier l'installation de ces applications sur les ressources qui les exécutent.

Il fournit une abstraction supplémentaire à la notion de requête DIET, en introduisant la notion de *jobs* comportant plusieurs *workunits*. Cette abstraction permet d'exploiter facilement le parallélisme de données. Toutes ces informations sont stockées dans une base de données relationnelle permettant de garder une trace des exécutions effectuées par la grille Décrypthon. De plus, il permet de gérer les erreurs au niveau d'une *workunit*. En effet il se peut qu'un site soit rendu indisponible au cours d'une opération de maintenance. Dans ce cas une nouvelle requête DIET appelant le service pour cette workunit est effectuée. Ainsi, une *workunit* peut être liée à plusieurs requêtes DIET.

À la fin de l'exécution d'un *job* qui peut être composé de plusieurs *workunits*, c'est à dire plusieurs résultats de requêtes DIET, le DIET_Webboard se charge de récolter l'ensemble des résultats dans une seule et même archive afin de permettre à l'utilisateur de télécharger ses résultats. Un courriel de notification est envoyé à l'utilisateur pour l'avertir de la fin de son *job*.

Enfin, nous avons développé une interface de programmation (API) spécifique dans le DIET_Webboard permettant de construire un portail internet dédié pour un utilisateur. Il suffit alors d'écrire la page de soumission, qui demande les paramètres nécessaires à la soumission d'un *job* pour un service, le reste (i.e gestion des données ; authentification utilisateurs, paramètres du *job* et les données résultats) est automatiquement géré.

La conception d'un portail internet de haut niveau est donc très simple et se retrouve résumée à l'écriture d'un simple formulaire permettant à l'utilisateur d'entrer ses paramètres.

3.5.4 L'ordonnancement sur les ressources Décrypthon

Nous l'avons rappelé, les ressources Décrypthon sont gérées par un ordonnanceur local spécifique à chaque site. Il existe deux ordonnanceurs locaux différents : OAR et Loadleveler, la configuration locale est indépendante et différente pour les 5 sites.

Pour écrire le SeD DIET nous nous sommes servis de deux fonctionnalités de l'intergiciel DIET :

- le développement d'un ordonnanceur spécifique ;
- la faculté de soumettre à des ordonnanceurs locaux.

L'ordonnanceur spécifique permet de choisir le site qui va être utilisé pour l'exécution d'une requête en fonction de la charge du site. Nous avons écrit un *plug-in scheduler* qui interroge l'ordonnanceur local du site (OAR ou Loadleveler) pour obtenir les tâches actuellement en cours d'exécution ou en attente de ressources. Les deux ordonnanceurs ont une politique de type « FIFO avec backfilling » (une tâche peut doubler une autre uniquement si elle a la place de s'intercaler sans perturber celles déjà ordonnancées). Nous estimons alors le temps de terminaison d'une requête DIET à partir :

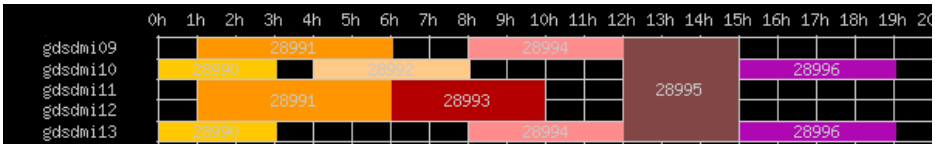
- de la durée d'exécution maximum de la requête (*walltime*) ;
- du diagramme de Gantt que nous reconstruisons à partir des travaux déjà en cours.

La figure 3.18 présente un exemple de calcul du temps de terminaison en fonction de la charge initiale de la plate-forme 3.18(a) pour 3 cas de figure :

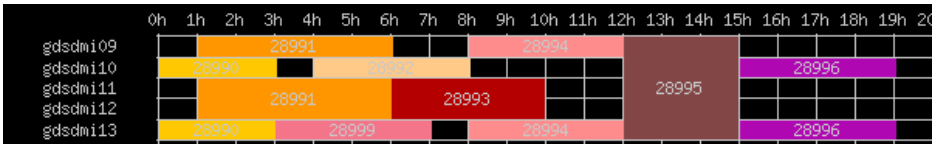
- 3.18(b) une requête demandant un processeur pour une durée de 4 heures, le temps de terminaison prévu est 7h.
- 3.18(c) une requête demandant deux processeurs pour une durée de 2 heures, le temps de terminaison prévu est 8h.
- 3.18(d) une requête demandant trois processeurs pour une durée de 4 heures, le temps de terminaison prévu est 19h.

Ce temps de terminaison relatif au temps de soumission EFT(*Earliest Finish Time*) est remonté dans la hiérarchie suivant le mécanisme que nous avons décrit dans la partie 3.4.2 décrivant les ordonnanceurs spécifiques dans DIET. Nous remontons aussi le TIMESINCELASTSOLVE. L'ordre de priorité pour sélectionner le site sur lequel sera soumis la requête est alors : priority_min(TIMESINCELASTSOLVE), priority_max(TIMESINCELASTSOLVE). Ce qui revient à sélectionner le site sur lequel la requête se terminera le plus tôt. Si les temps de terminaison sont identiques, nous choisissons alors le site sur lequel nous avons soumis il y a le plus longtemps.

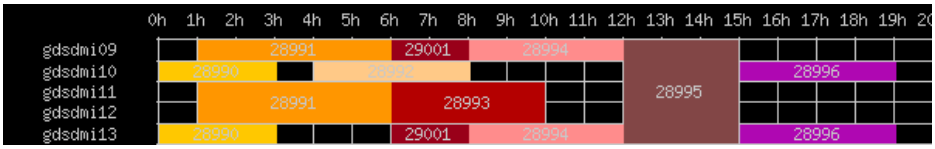
À terme nous prévoyons d'inclure cette fonctionnalité dans un collecteur d'informations spécifique aux ordonnanceurs locaux, que l'on nommera :



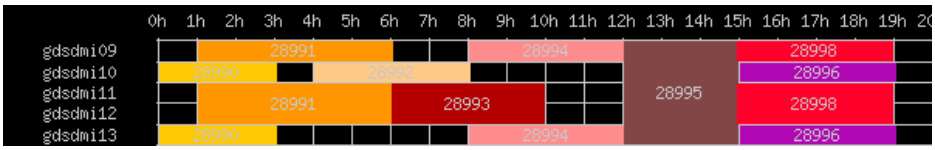
(a) Charge initial de l'ordonnanceur ;



(b) Charge initiale + 1 tâche : nb_proc=1, *walltime*= 4 h ;



(c) Charge initiale + 1 tâche : nb_proc=2, *walltime*= 2 h ;



(d) Charge initiale + 1 tâche : nb_proc=3, *walltime*= 4 h ;

Figure 3.18 – Prédiction du temps de terminaison dans les ordonnanceurs batch.

CoRI-batch.

Une fois le site sélectionné, une requête est soumise sur le SeD qui soumet l'application à l'ordonnanceur local du centre de calcul. Lorsque l'application est effectivement lancée sur une ressource du centre de calcul, nous notifions le commencement des calculs au DIET_Webboard. À la fin des calculs, les résultats sont envoyés sous forme d'une archive résultat sur un site de stockage, et nous notifions le DIET_Webboard avec les informations concernant les ressources consommées (taille des résultats et temps processeur nécessaire pour le calcul). Toutes ces informations sont stockées dans la base de données du DIET_Webboard.

La figure 3.19 illustre l'implantation fait du SeD DIET Décrypthon sur chaque site de la grille Décrypthon.

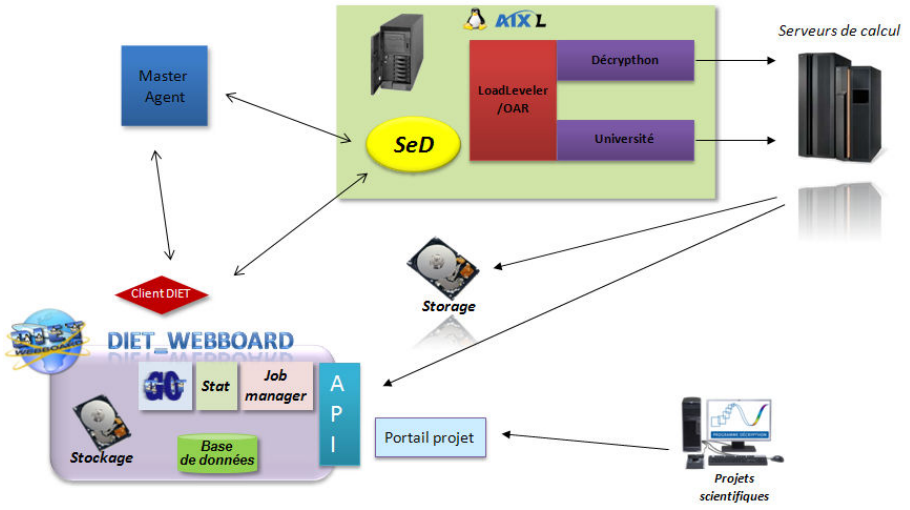


Figure 3.19 – Fonctionnement de DIET avec les ordonnanceurs locaux des centres de calcul.

Discussions

Le DIET_Webboard offre un portail Web d'administration et de gestion d'une grille adressée par l'intergiciel DIET. Il fournit le tableau de commande des ressources de la grille et l'abstraction nécessaire pour permettre un accès transparent à l'administrateur comme à l'utilisateur.

De plus, le DIET_Webboard ajoute les fonctionnalités de sécurité basiques qui manquent à DIET. Ainsi, les utilisateurs s'identifient au niveau du portail en utilisant un identifiant et un mot de passe. La gestion de leurs droits est alors prise en charge par le portail Web en définissant des droits d'accès aux différentes applications et données accessibles depuis les ressources Décrypthon.

Le DIET_Webboard a été développé pour le Décrypthon. Cependant, il n'est pas limité au cadre de ce projet. Nous l'avons développé comme un outil complémentaire à DIET, destiné à piloter n'importe quelle grille de ressources orchestrée par l'intergiciel DIET. Aussi, ce portail Web pourrait être utilisé par d'autres projets s'appuyant déjà sur l'intergiciel DIET comme le projet Grid-TLSE [5].

3.5.5 Les applications gridifiées

Actuellement trois applications sont en production dans la grille Décrypthon :

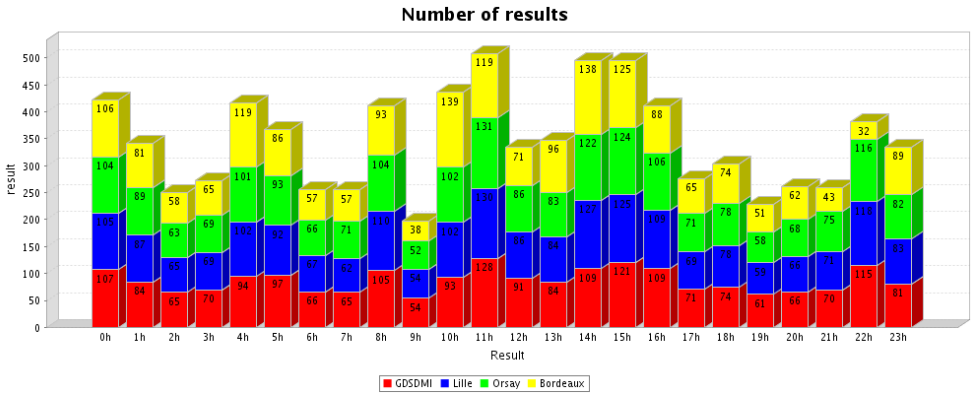


Figure 3.20 – Exemple d'un jour d'activité sur les différents sites plate-forme Décrypthon.

- l'application PipeAlign et MACSIMS du projet MS2PH (voir la section 2.3.1) ;
- l'application SpikeOmatic (voir la section 2.3.4) ;
- l'application d'amarrage moléculaire du projet HCMD (voir la section 2.3.3).

Applications du projet MS2PH

L'application MS2PH est constituée d'une suite de programmes d'analyse de séquences, qui peuvent également être utilisés indépendamment les uns des autres. La construction de l'alignement multiple est réalisée à partir de la séquence soumise par l'utilisateur, en s'appuyant sur une version ajustée de *PipeAlign* [74] (voir figure 3.21). L'alignement retenu est ensuite annoté par l'intermédiaire de MACSIMS qui s'adresse directement à un serveur de banque de données [71]. Cette application est lancée par le SeD DIET sous la forme d'un script *tcl* qui lance et gère les différents binaires. Il prend une série de paramètres en entrée et a besoin de banques de données (fichiers plats) pré-installées sur les sites. Au besoin, elles sont mises à jour.

La sortie finale de la suite de programme *PipeAlign* est un MACS (**M**ultiple **A**lignement of **C**omplete **S**equences) validé de haute qualité et une annotation au format XML de cet alignement grâce au programme MACSIMS.

Si l'utilisateur fournit un fichier contenant plusieurs séquences d'acides aminés, celles-ci sont automatiquement séparées en plusieurs fichiers. Cette séparation permet d'exploiter le parallélisme de données en lançant une *workunit* par séquence. Aujourd'hui les travaux soumis par les chercheurs com-

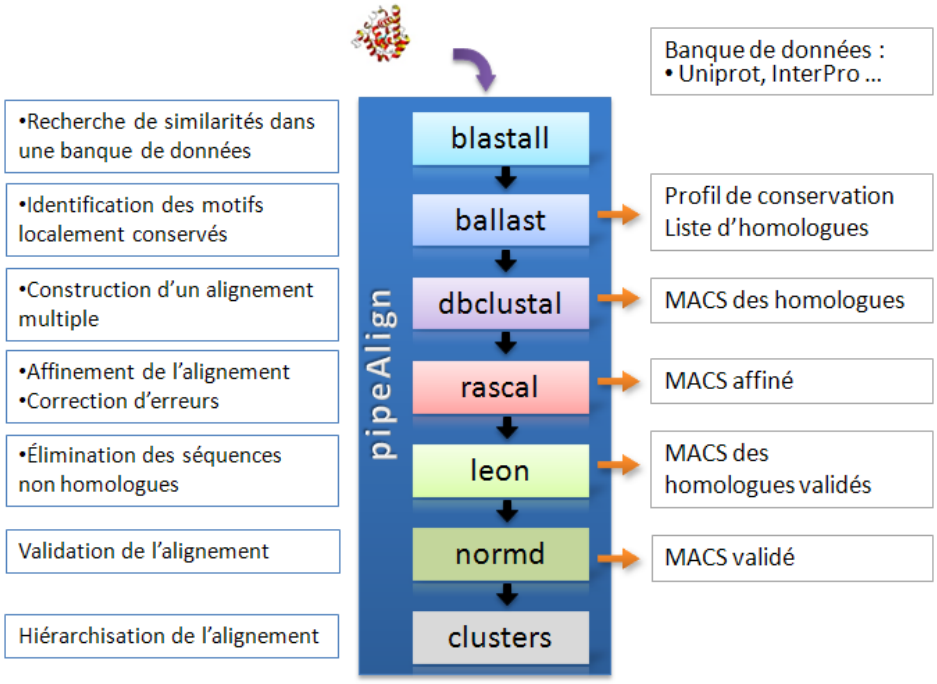


Figure 3.21 – Enchaînement des programmes composants le PipeAlign.

portent environ 5000 séquences d'acides aminés. Les résultats sont réunis automatiquement par le serveur de grille offrant à l'utilisateur la possibilité de télécharger les 5000 MACS et annotations en une seule fois.

L'application MAXDo

L'application MAXDo est un programme d'amarrage moléculaire protéine-protéine écrit en FORTRAN. Il explore les sites d'interaction entre un récepteur fixe avec un ligand mobile. Les paramètres de l'exploration sont découpés automatiquement en petites tranches d'exploration qui sont autant de *workunits*. Nous le verrons dans le chapitre suivant, cette application est très gourmande en temps de calcul et génère peu de données. Elle est disponible (voir figure 3.22) sur la grille universitaire Décrypthon dans le but de tester et de valider les différents jeux de paramètres [81] avant le lancement d'une campagne de calcul sur la grille de volontaires du World Community Grid.

Cette application est surtout destinée à être utilisée dans l'environnement du World Community Grid qui offre des moyens de calcul bien plus intéressants que la grille universitaire Décrypthon pour ce type d'application.

Figure 3.22 – Capture d'écran de la page de soumission MAXDo.

L'application SpikeOmatic

L'application SpikeOmatic est un programme d'analyse d'enregistrements issus d'électrodes recevant des signaux électriques (potentiels d'actions). Le programme, écrit dans le langage *R* [77], travaille sur les données brutes acquises au cours d'une expérience ou d'un examen. Il tente, à partir de modèles physiques et de méthodes statistiques, d'identifier les neurones responsables de l'activité électrique détectée par l'électrode.

Le portage (voir figure 3.23) de cette application sur les ressources de la grille Décrypthon permet aux chercheurs de valider plusieurs modèles d'analyse et de lancer leur programme sur plusieurs enregistrements en parallèle.

Les utilisateurs fournissent les paramètres d'analyse et l'adresse des enregistrements préalablement sauvegardés sur un dépôt accessible par les ressources de calcul. Les résultats sont fournis sous la forme d'une archive qu'ils peuvent examiner sans avoir eu à faire les calculs sur leur propre machine. La durée des calculs dépend des paramètres et de la durée de l'enregistrement à analyser.

Figure 3.23 – Capture d'écran de la page de soumission SpikeOmatic.

Les autres applications.
L'application du projet d'expression de l'évolution des gènes a été mise en place dans la première version de la grille Décrypthon. L'application du projet « Défauts d'épissage et maladies génétiques » est en cours d'adaptation et de portage, il est important de noter que cette application est un programme parallèle MPI s'exécutant sur plusieurs processeurs (4 à 16 processeurs).

Enfin deux applications des deux derniers projets décrits dans le chapitre précédent seront portées d'ici la fin de l'année.

Parmi les applications qui fonctionnent sur la grille Décrypthon, les périodes d'activités sont distinctes et dépendent des avancements respectifs des projets. De temps en temps, les binaires, les bibliothèques et les scripts des applications sont mis à jour, suivant l'évolution des projets scientifiques.

Il existe un travail récurrent pour le projet MS2PH qui est lancé de manière périodique. Tous les deux mois 1000 séquences de protéines impliquées dans les maladies génétiques humaines sont alignées et annotées. Les résultats sont directement envoyés sur un serveur de stockage afin de mettre à jour la base de données MS2PH-db [71].

3.6 Conclusion

La grille Décrypthon composée de 6 sites universitaires est née du désir des acteurs du programme Décrypthon d'apporter une plate-forme de production permettant aux projets sélectionnés d'utiliser un supplément de ressources qu'ils n'ont pas la possibilité ou les moyens d'héberger. La plate-forme a d'abord été gérée avec succès avec l'intergiciel GridMP de la société *United Device*, puis au cours d'une phase de migration, nous avons fait évoluer la plate-forme vers l'intergiciel DIET développé par l'équipe GRAAL de l'ENS-lyon.

La période de transition a permis le développement de DIET_Webboard comme portail d'accès et de gestion de la plate-forme universitaire. Cette évolution n'a rien changé pour les utilisateurs des projets scientifiques. DIET est, quant à lui, passé du statut d'intergiciel de recherche au statut d'intergiciel employé en production grâce à ses qualités, sa robusteuse, sa modularité et son panel d'outils disponibles. De plus, notons que le DIET_Webboard, développé dans le cadre du Décrypthon, est désormais disponible en tant que portail indépendant permettant de gérer n'importe quelle grille dont les ressources sont pilotées par l'intergiciel DIET.

Le DIET_Webboard apporte à DIET l'identification et la sécurité d'accès qui lui faisait défaut jusqu'à présent. Cette couche supplémentaire permet de conserver les propriétés de performance que nous montrerons dans le chapitre suivant à travers des expériences dimensionnantes sur plus de 1000 processeurs qui ont été menées sur la plate-forme de recherche Grid'5000.

Chapitre 4

Expériences dimensionnantes sur grilles informatiques

Sommaire

4.1 Introduction

Dans ce chapitre, nous présentons un ensemble de cas d'utilisation de grilles, allant de la grille de recherche Grid'5000 jusqu'à la grille de volontaires World Community Grid, en passant par la grille Décrypthon. Nous montrons sur des cas concrets d'utilisation comment la grille devient un outil informatique indispensable afin d'obtenir des résultats ou de valider des fonctionnalités.

En premier lieu, nous présenterons diverses expériences réalisées sur la grille de recherche Grid'5000. Dans cette section, toutes les expériences sont en rapport avec l'intergiciel DIET que nous avons présenté dans la section 3.4 du chapitre précédent. L'objectif est de valider certaines fonctionnalités de l'intergiciel dans un environnement de grande taille. Nous décrirons les modes opératoires, les étapes de travail, puis les résultats obtenus. Nous aborderons enfin les enseignements issus de ces expériences.

En second lieu, nous exposerons un travail complet de préparation fait sur l'application d'amarrage moléculaire du projet HCMD soutenue par le Décrypthon. Nous dévoilerons les étapes qui ont précédé le lancement sur la grille de volontaires Word Community Grid. Nous montrerons comment deux grilles, l'une dédiée et l'autre volatile, peuvent être utilisées de manière complémentaire pour préparer des calculs qui auraient demandé plus de 8000 ans sur une seule machine.

En dernier lieu, nous établirons un point de comparaison entre une grille dédiée et une grille de volontaires qui *a priori* semblent opposées, de par leurs architectures et leur modèle de fonctionnement. À partir de la notion de processeurs virtuels à plein temps et de processeurs dédiés de référence nous donnerons un facteur de conversion entre les deux grilles. Nous emploierons cette notion pour estimer les besoins de la deuxième phase de calcul du projet HCMD.

4.2 Diverses expériences avec DIET

Dans cette section, nous allons présenter plusieurs expériences que nous avons réalisées sur la grille de recherche Grid'5000 avec l'intergiciel DIET. L'architecture générale de l'intergiciel DIET a été présentée dans le chapitre précédent.

Nous décrivons ici les protocoles d'expériences mis en place et les différents obstacles qui les ont jalonnés. Nous mettons en lumière ce que nous avons appris au cours de ces expériences non seulement riches en enseignements mais aussi à l'origine d'améliorations et de création d'outils dans l'écosystème

autour de DIET.

Pour commencer, il est important de préciser qu'invariablement les étapes des expériences sur la grille de recherche Grid'5000 avec l'intergiciel DIET sont les suivantes :

- définition d'un protocole d'expériences et d'un objectif ;
- réservation d'un ensemble de machines sur Grid'5000 ;
- déploiement d'une plate-forme/hiérarchie DIET sur les machines réservées ;
- lancement des tests ;
- récolte et analyse des données issues de l'expérience.

Nous avons choisi de présenter deux expériences différentes qui utilisent les fonctionnalités offertes par la grille de recherche Grid'5000 et qui donnent un aperçu des possibilités de l'intergiciel DIET.

Pour toutes les expériences nous n'utilisons pas une image système sur laquelle est installée l'intergiciel DIET. Plusieurs raisons peuvent être avancées. D'une part, l'intergiciel DIET ne nécessite pas de privilèges utilisateurs particuliers pour être exécuté, un simple compte utilisateur et un accès à la machine suffisent. D'autre part, l'utilisation d'une image système ajoute à chaque expérience un délai de redémarrage des machines. Enfin, au moment des expériences, tous les sites n'offraient pas encore cette fonctionnalité, il n'y avait donc aucune raison d'avoir recours à une image système et à un déploiement via l'outil Kadeploy [103].

Nous avons donc au préalable installé sur notre compte utilisateur Grid'5000, les logiciels nécessaires au déploiement de DIET, à savoir omniORB, LogService, GoDIET, DIET et tous les binaires ou librairies utiles au fonctionnement des services déployés dans la hiérarchie.

4.2.1 Passage à l'échelle Grid'5000 de DIET

C'est une des expériences qui a demandé le plus d'efforts et qui a été à la genèse de plusieurs outils facilitateurs d'expériences de ce type. Le but avoué de cette série d'expériences était de réaliser un déploiement record d'une hiérarchie DIET en utilisant l'ensemble des sites et des machines disponibles au moment de l'expérience.

Réservation de tout Grid'5000

Pour cette expérience, qui regroupe plusieurs essais, nous avons d'abord dû faire face aux difficultés d'utilisation de Grid'5000. Ces expériences se sont déroulées au début de l'année 2006, Grid'5000 en était encore à ses

débuts, certains sites venaient juste de rejoindre le projet et de recevoir leurs machines. Beaucoup de problèmes de configuration sur les différents sites rendaient l'utilisation conjointe des 8 sites alors disponibles assez lourde. Au niveau du site de Rennes, deux outils de réservation coexistaient : GridPrem's et OAR.

Il était techniquement impossible de faire une réservation sur tous les sites via l'outil OARgrid. Nous devions nécessairement synchroniser une réservation localement sur chaque grappe de machines afin d'avoir un ensemble de ressources sur tous les sites. Il n'était pas rare de faire des réservations, et lorsque celles-ci étaient actives, d'avoir des machines qui ne répondaient pas ou dont l'accès nous était impossible.

Voici un exemple de la sortie des commandes *oarsub* pour la réservation de 598 machines réparties sur 7 sites :

```
SOPHIA :
oarsub -r "2006-02-10 13:00:00" -l nodes=90,walltime="6:00:00"
Host:Port = frontale.sophia.grid5000.fr:60078
 IdJob = 115628
Reservation mode : waiting validation
Reservation valid --> OK

TOULOUSE :
oarsub -r "2006-02-10 13:00:00" -l nodes=50,walltime="6:00:00"
Host:Port = cict-254.toulouse.grid5000.fr:55098
 IdJob = 24309
Reservation mode : waiting validation
Reservation valid --> OK

LILLE :
oarsub -r "2006-02-10 13:00:00" -l nodes=45,walltime="6:00:00"
Host:Port = frontale:53940
 IdJob = 748
Reservation mode : waiting validation
Reservation valid --> OK

BORDEAUX :
oarsub -r "2006-02-10 13:00:00" -l nodes=45,walltime="6:00:00"
Host:Port = frontale.bordeaux.grid5000.fr:36157
 IdJob = 12972
Reservation mode : waiting validation
Reservation valid --> OK
```

```
LYON :
oarsub -r "2006-02-10 13:00:00" -l nodes=50,walltime="6:00:00"
Host:Port = capricorne.lyon.grid5000.fr:35795
 IdJob = 25447
Reservation mode : waiting validation
Reservation valid --> OK

ORSAY :
oarsub -r "2006-02-10 13:00:00" -l nodes=190,walltime="6:00:00"
Host:Port = devgdx002.orsay.grid5000.fr:40465
 IdJob = 29165
Reservation mode : waiting validation
Reservation valid --> OK

RENNES :
Utilisation de l'interface de réservation GridPrem's
réservation de 2 fois 64 machines.
```

Cette réservation concernait un total de 598 machines sur 7 des 8 sites disponibles à l'époque. Le site de Grenoble n'a pas été utilisé en raison du faible nombre de machines installées. Cette réservation représentait près de 80% des machines alors présentes dans la grille Grid'5000. Une fois les réservations faites, nous avons vérifié que les machines obtenues étaient effectivement disponibles et utilisables : à l'aide d'un script nous nous sommes connectés à la liste des 598 machines et nous avons éliminé celles qui ne répondaient pas.

Déploiement de la hiérarchie DIET

À ce stade, nous avions une liste de machines réservées sur lesquelles nous pouvions déployer une hiérarchie DIET. L'outil nommé GoDIET [24] permet son déploiement en respectant l'ordre imposé par la hiérarchie. L'utilitaire GoDIET nécessite un fichier décrivant l'ensemble des ressources sur lesquelles il doit déployer la hiérarchie DIET, ainsi que la description explicite de celle-ci.

Étant donné que les machines obtenues pour une réservation ne sont connues qu'au moment de la réservation, nous étions obligés de générer le fichier de description de notre déploiement en début de réservation.

La figure 4.1 montre une capture d'écran d'un déploiement réalisé au cours de ces expériences. Cette hiérarchie comportait : 1 *MasterAgent*, 8

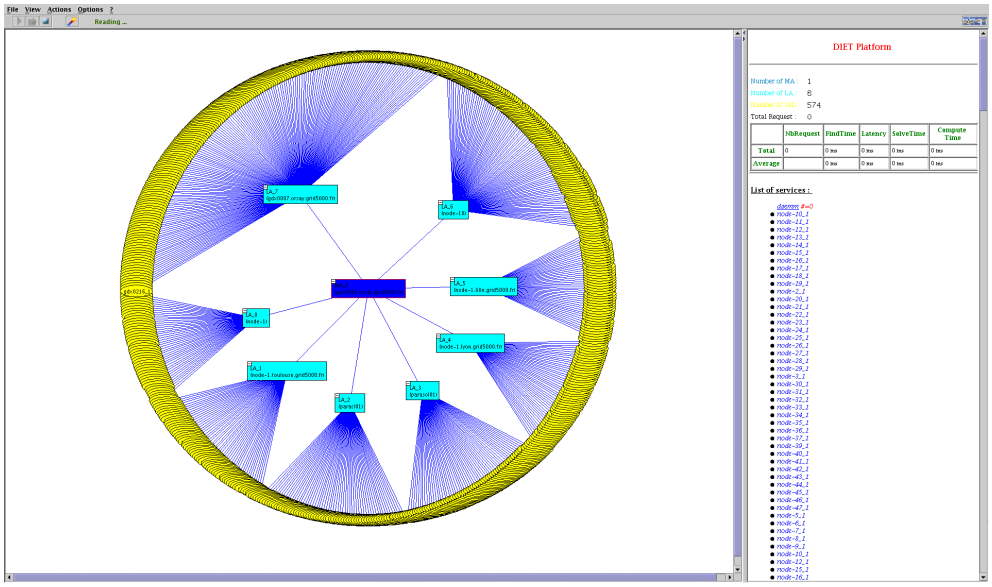


Figure 4.1 – Capture d'écran de VizDIET d'un déploiement sur 583 machines.

LocalAgents sur 7 sites Grid'5000. Chaque *LocalAgent* était localisé sur une grappe de machines. La hiérarchie comportait 574 SeDs disposant du service DGEMM[1]. Le service de surveillance (LogService) était activé. Dans le détail, nous avons utilisé les sites de Bordeaux (44 machines), Lille (44 machines), Lyon (50 machines), Orsay (178 machines), Sophia (90 machines), Rennes (2 fois 61 machines), Toulouse (55 machines) : soit un total de 583 machines sur les 598 machines bi-processeurs demandées, c'est-à-dire 1166 processeurs.

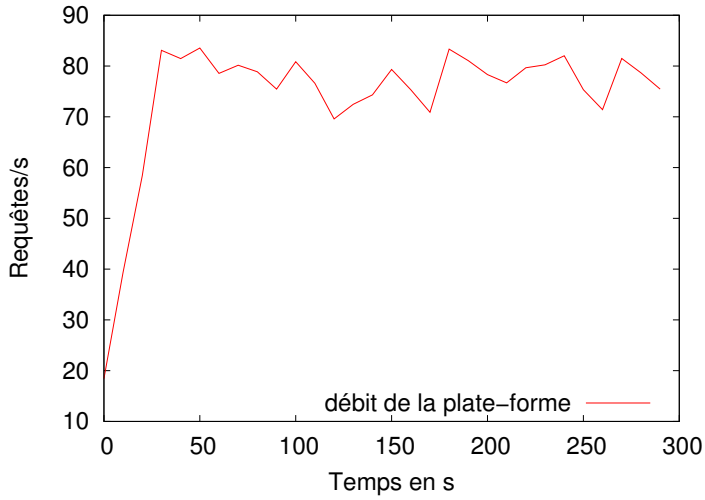
Test et utilisation de la plate-forme

À la suite du déploiement ainsi réalisé, nous avons lancé une série de 10 clients depuis 112 machines différentes faisant des requêtes à la plate-forme DIET. Les 1120 clients ont été exécutés deux par deux toutes les 20 secondes sur une période de 9 min et 20 s. Les clients appellaient le service DGEMM pour deux matrices de petites tailles (10x10) et attendaient le résultat avant de soumettre à nouveau, une requête.

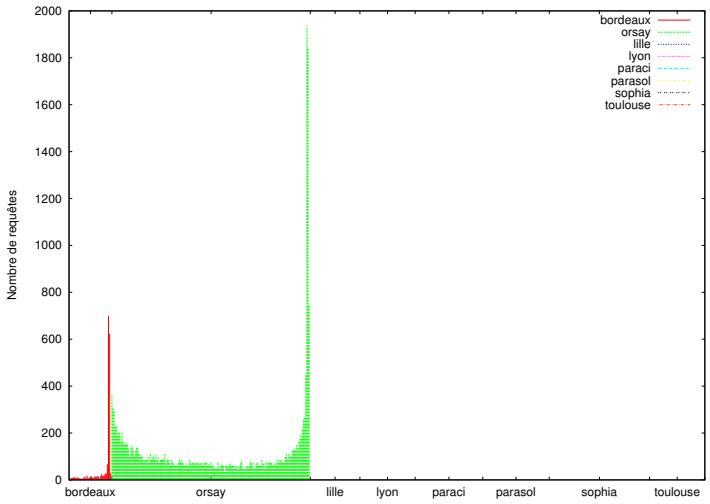
Le but était de tester le bon fonctionnement de la hiérarchie, et d'observer le débit de la plate-forme, c'est à dire le nombre de requêtes traitées par seconde. Ce débit est calculé au niveau du *MasterAgent*.

La stratégie d'ordonnancement mise en place était celle par défaut : le

[1]service de multiplication de matrice



(a) Débit de la plate-forme ;



(b) Répartition inégale des requêtes ;

Figure 4.2 – plate-forme DIET composée de 1 MA, 8 LA et 574 SeDs répartis sur 8 sites Grid'5000.

SeD choisi est celui qui a exécuté une requête il y a le plus longtemps. Ce qui revient à faire un quasi *round-robin* sur toutes les machines (cf. section 3.4).

L'enregistrement du débit est fait sur une durée de 5 minutes, 23 274 requêtes clients ont été traitées. Nous observons sur la figure 4.2(a) que le débit de la plate-forme a atteint rapidement un plafond autour de 75 requêtes/s. Ce qui était peu compte tenu du nombre de clients mis en jeu dans l'expé-

rience. Nous avons donc cherché à comprendre pourquoi le débit n'était pas plus important.

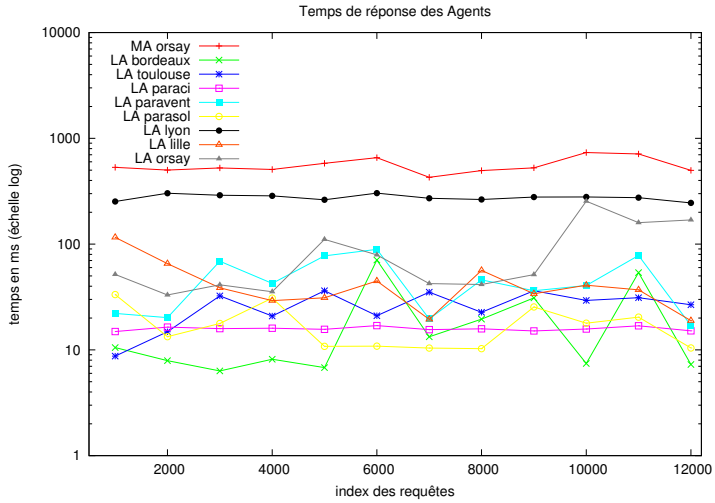
En analysant plus finement les traces de l'expérience, il est apparu que la charge des requêtes était répartie uniquement sur deux grappes de machines : les 178 *SeDs* situés sur le site d'Orsay et les 44 *SeDs* situés sur le site de Bordeaux. Après investigation, nous avons découvert un problème de compatibilité entre DIET et deux versions d'omniORB (4.0.5, 4.0.6), en effet les deux sites d'Orsay et de Bordeaux étaient installés avec la version 4.0.5, et 4.0.6 pour les autres. Ceci a entraîné un problème dans le choix des machines qui était restreint aux seuls sites d'Orsay et de Bordeaux. La figure 4.2(b) montre la répartition des requêtes sur les 7 sites impliqués dans l'expérience.

Il apparaît aussi, qu'indépendamment des versions d'omniORB, la charge est inégalement répartie sur les machines. Ce problème est directement lié à l'ordonnanceur par défaut de DIET qui choisit le plus vieux SeD ayant résolu une requête. Dans la mesure où plusieurs clients effectuent des appels en parallèle sur la plate-forme, si ceux-ci sont traités par le *Master Agent* avant que les clients aient commencé la résolution du service, alors le même SeD est toujours choisi.

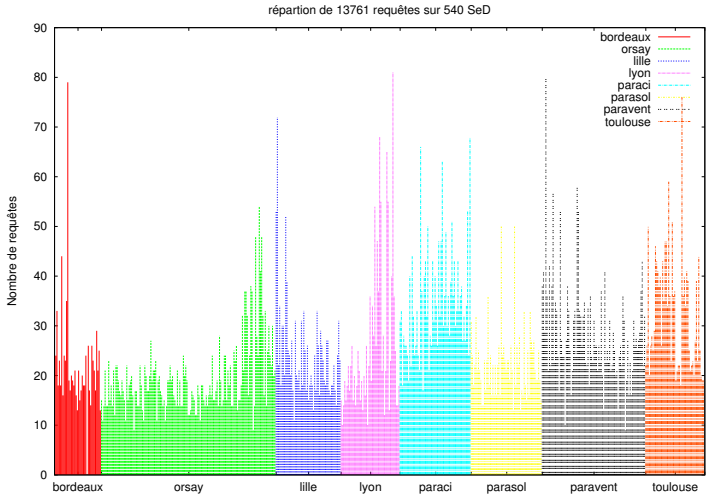
Dans la configuration de notre expérience, nous étions typiquement dans ce cas de figure : jusqu'à 1120 clients effectuaient des requêtes en parallèle sur la plate-forme. Ce problème paraît inévitable étant donné le fonctionnement en deux étapes de DIET et le caractère réparti de l'ordonnancement. Cependant, nous avons tout de même amélioré ce fonctionnement en mettant à jour au plus tôt l'information stockée dans le SeD sur la date de début de résolution d'un service, afin de réduire au minimum le temps en dessous duquel nous pouvons garantir un équilibrage de charge.

Après correction des problèmes nous avons refait une expérience similaire dans le but de quantifier les temps de réponse d'une plate-forme DIET comportant un grand nombre de SeDs répartis sur plusieurs sites.

La figure 4.3(b) montre, cette fois, une répartition plus uniforme des requêtes sur l'ensemble des SeDs disponibles. Cependant la charge reste inégale. Étant donné le caractère réparti des clients et le mode de fonctionnement de DIET, il n'existe pas de moyens pour résoudre ce problème. Néanmoins, celui-ci ne survient que lorsque des requêtes sont soumises à une fréquence plus rapide que le temps de réponse de la hiérarchie. La figure 4.3(a) montre la courbe des temps de réponse des agents de la plate-forme. La réponse du *Master Agent* dépend de celle de tous ces fils : le temps de réponse de la hiérarchie est donc donné par celui-ci. Au cours de cette expérience, le temps moyen de réponse du *Master Agent* pour les 12 048 requêtes était de 0,558 s, avec un écart type de 0,376 s.



(a) Temps de réponse des Agents ;



(b) Répartition des requêtes ;

Figure 4.3 – plate-forme DIET composée de 1 MA, 8 LA et 540 SeDs répartis sur 8 site Grid'5000.

Enseignements tirés des expériences

Au cours de ces expériences dimensionnantes, où l'objectif affiché était de tester le passage à l'échelle de l'intergiciel DIET, nous avons appris beaucoup de choses.

Tout d'abord, il s'est avéré long et fastidieux de mettre en place ce genre d'expériences avec les outils disponibles dans Grid'5000. On peut

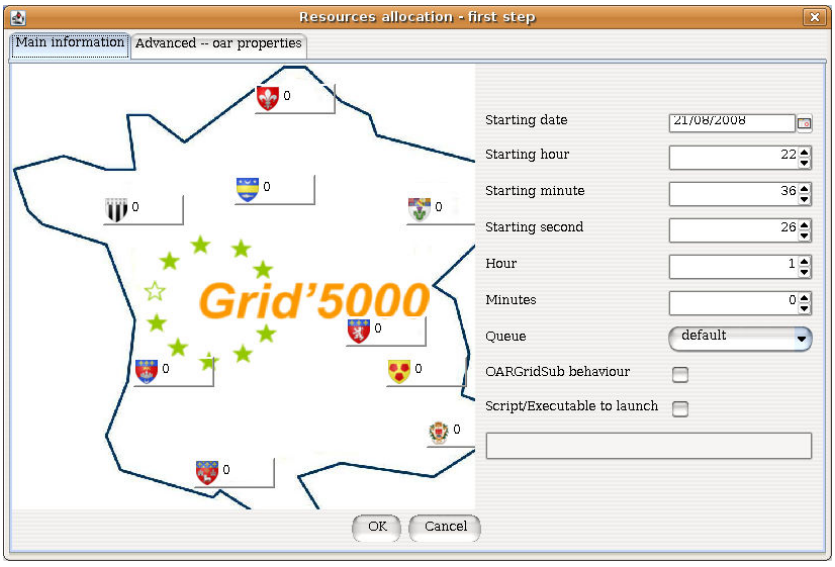


Figure 4.4 – Outils de réservation GRUDU.

avancer le fait que la grille Grid'5000 avait, à cette époque, une gestion très localisée des sites sans avoir une politique globale d'homogénéisation des environnements afin de faciliter les expériences multi-sites. Une réponse donnée à ces difficultés a été le développement de l'outil GRUDU[1]. Cet outil graphique (voir figure 4.4), conçu par l'équipe GRAAL, permet de faire des réservations multi-sites et de récupérer l'ensemble des nœuds de la réservation.

Une deuxième difficulté, cette fois liée à DIET et son utilitaire de lancement GoDIET. Le déploiement de DIET est décrit de manière statique dans un fichier XML. Étant donné le caractère dynamique des machines attribuées pour une réservation, il est primordial d'avoir un outil permettant de générer automatiquement la hiérarchie que l'on désire déployer à partir des machines réellement obtenues.

C'est pourquoi l'outil *XmlGoDIETGenerator* (figure 4.5) a été développé. Il permet, à partir d'un ensemble de réservations et des machines associées, de générer le fichier XML GoDIET correspondant. Ses propres modèles de hiérarchie peuvent être définis : étoile, hiérarchie à 2 étages avec 1 MA et 1 LA sur chaque grappe de machines, *etc.*

[1]GRUDU : ***G**rid'5000 **R**eservation **U**tility for **D**eployment **U**sage*

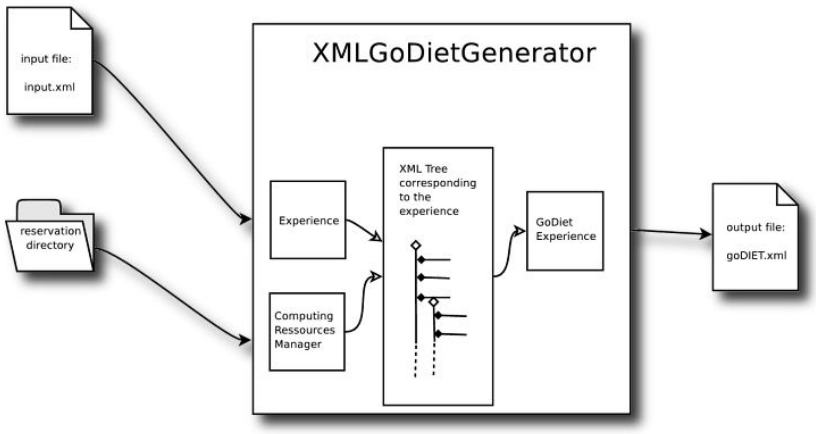


Figure 4.5 – *XmlGoDIETGenerator* : outils de génération de configuration GoDIET.

Enfin, ces expériences ont permis de délimiter certaines fonctionnalités de l'intergiciel DIET, et de mettre en avant sa capacité à passer à l'échelle d'une grille possédant plus de 1000 processeurs. Il est certain que ces expériences contribuent au renforcement de l'image de DIET comme un intergiciel modulaire et léger capable de s'adapter à l'environnement d'une grille.

4.2.2 Outil de surveillance pour une application distribuée

Comprendre le comportement d'une application distribuée est un problème difficile. Les grilles de calcul ajoutent un niveau de difficulté supplémentaire lorsque les applications distribuées sont réparties sur un ensemble d'ordinateurs dans des domaines administratifs disparates. Les développeurs d'intergiciels pour les grilles tentent d'optimiser les performances des applications en tenant compte de ces problèmes. Ils ont besoin d'un outil simple pour surveiller et connaître le comportement de leur intergiciel.

Un système complet de surveillance (LogService) à la fois robuste et léger a été développé afin de refléter le comportement d'une plate-forme hautement distribuée. Par la suite nous rappellerons comment le LogService a été intégré aux éléments de la hiérarchie DIET. Nous présenterons un modèle du coût de cette intégration et quelques résultats expérimentaux montrant les surcoûts engendrés par le LogService sur la hiérarchie DIET.

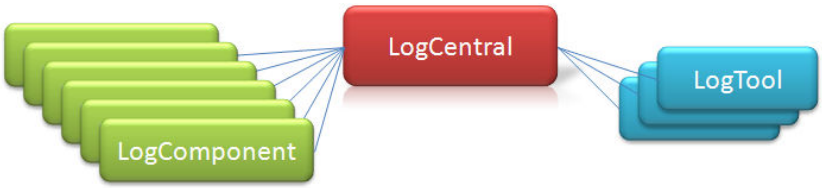


Figure 4.6 – Les éléments du LogService.

Le LogService et DIET

Comme décrit dans la section 3.4.3, le LogService [14] est un système de surveillance. Il relaye les informations qui surviennent dans un environnement distribué. Ce service utilise la technologie CORBA et il s'interface avec une application existante désirant mettre en place un système d'information. Chaque élément de l'application dont le développeur veut surveiller le comportement, se voit attacher un *LogComponent* qui relate les événements à un composant centralisé : le *LogCentral*. Celui-ci stocke, ou transmet l'information aux *LogTools*. Les composants *LogTools* sont chargés, quant à eux, d'interpréter les messages (figure 4.6).

Le Logservice définit et implante plusieurs fonctionnalités :

- **un mécanisme de filtrage** qui permet de réduire le nombre de messages qui transitent par le réseau. Chaque capteur (*LogComponent*) n'envoie que les messages utiles et chaque outil d'analyse *LogTool* ne reçoit que les messages dont il a la charge d'analyser.
- **un mécanisme d'horodatage des évènements** qui estampille les messages grâce à l'horloge globale du *LogCentral*. Comme l'horloge système de chaque hôte peut être différente, le *LogCentral* mesure le décalage et corrige la date des messages qu'il reçoit avant de les retransmettre.
- **une mise en attente des messages**. Les *LogComponents* peuvent introduire un délai avant de retransmettre les messages vers le LogCentral.
- **un réordonnancement des messages**. Le LogCentral dispose lui aussi d'un mécanisme de mise en attente des messages reçus des *LogComponents* avant de les retransmettre aux *LogTools*. Cette mise en attente peut lui permettre de remettre les messages dans l'ordre chronologique avant de les retransmettre.
- **la surveillance des éléments**. Chaque *LogComponent* peut se connecter et disparaître au cours du temps. L'état du système est en permanence surveillé par le *LogCentral* qui vérifie la présence des *Log-*

Components déclarés.

– **différents type de messages**. Par défaut le LogService définit 2 types de messages :

Les messages dits volatils : ce sont tous les messages reçus par le LogCentral. Afin de ne pas encombrer la mémoire du LogCentral, aucun de ces messages n'est stocké. Ils sont transmis uniquement aux *LogTools*, libre à eux de les interpréter et de les stocker.

Les messages dits permanents : ce sont les messages que le *LogCentral* doit garder en mémoire afin de les transmettre à chaque nouveau *LogTool*. Par défaut l'enregistrement d'un nouveau *LogComponent* est un message permanent.

L'intégration du LogService dans l'architecture DIET est décrite dans le chapitre 3. Elle permet de suivre l'activité des éléments de la hiérarchie en fonction des événements qui surviennent.

Nous avons décrit le fonctionnement détaillé de DIET (cf. section 3.4.2). Schématiquement, il y a deux étapes principales lorsqu'un client appelle un service DIET : la **recherche du service** par les agents (modélisée par une *findRequest*) et **l'invocation du service** (*SolveRequest*). Une requête *DIETResquest* est donc la concaténation des deux requêtes précédentes. Enfin, un dernier objet, nommé *DIETLogEvent*, caractérise un message/événement atomique dans l'environnement. Les messages *DIETLogEvent* sont séparés en deux catégories : des messages d'**état et de configuration** et des messages d'**activité et d'information**. Voici la liste des messages regroupés en fonction des deux catégories :

◦ messages de configuration de la plate-forme DIET :
  – IN : arrivée d'un nouvel élément de la hiérarchie DIET, cela peut être un MA, MA_{DAG}, LA ou un SeD ;
  – OUT : arrêt ou départ d'un élément. Ce message n'est pas forcément envoyé par l'élément, cependant LogCentral le génère automatiquement si l'élément ne répond plus ;
  – ADD_SERVICE : ajout d'un service dans la hiérarchie.

◦ messages d'activités et d'information :
  – ASK_FOR_SED : demande d'un SeD par un client pour exécuter un service ;
  – SED_CHOSEN : réponse des agents à la demande d'un client ;
  – BEGIN_SOLVE : début de l'invocation d'un service sur un SeD ;
  – END_SOLVE : fin de l'invocation d'un service sur un SeD ;
  – DATA_STORED : déclaration d'une donnée persistante dans la hiérarchie
  – DATA_RELEASED : destruction d'une donnée persistante dans la hiérarchie.

- DATA_TRANSFER_BEGIN : début d'un transfert d'une donnée persistante entre SeD ;
- DATA_TRANSFER_END : fin du transfert d'une donnée persistante ;
- JUXMEM_DATA_STORE : déclaration d'une donnée dans le système JuxMem[2] ;
- JUXMEM_DATA_USE : utilisation d'une donnée issue du système JuxMem ;
- DAG : déclaration de l'exécution d'un graphe de tâches par le MA_{DAG}.

En plus de l'intégration des *DIETLogComponents* à chaque élément DIET, il existe aussi un *DIETLogTool* qui permet d'enregistrer dans un fichier l'activité de la hiérarchie. Ce fichier peut alors être analysé par un outil graphique VizDIET, ou traité indépendamment. VizDIET peut aussi directement se connecter à *LogCentral*, afin d'avoir un rendu de l'activité en temps réel.

Modélisation du nombre de messages et expériences

Nous avons modélisé, avec une formule, le nombre de messages générés par le système de surveillance LogService mis en place dans DIET.

Soit $Nb_{log}(req_{serv})$ le nombre de requêtes pour un service *serv*, Nb_{act} est le nombre de messages correspondant à des messages d'activité dans la plate-forme, Nb_{SeD} est le nombre de SeD, $Nb_{SeD}(serv)$ est le nombre de SeD exécutant le service *serv*, $Nb_{act}(serv)$ est le nombre de messages d'activité pour le service *serv* et $Nb_{agent}(serv)$ est le nombre d'agents qui possèdent un SeD exécutant le service *serv* dans son sous-arbre. Enfin req_{serv} le nombre de requêtes pour le service *serv*. On a alors :

$$\begin{aligned} Nb_{log}(req_{serv}) &= Nb_{desc} + req_{serv}.Nb_{act} \\ Nb_{desc} &= Nb_{Agent} + Nb_{SeD} + \sum_{serv}(Nb_{SeD}(serv)) \\ Nb_{act} &= \sum_{serv}(Nb_{act}(serv)) \\ Nb_{act}(serv) &= 2.(Nb_{agent}(serv) + 1) \end{aligned}$$

Cette formule se simplifie si l'on considère maintenant que la plate-forme n'a qu'un seul service par SeD et que tous les SeD peuvent exécuter le service.

$$\begin{aligned} Nb_{log}(req) &= Nb_{desc} + req.Nb_{act} \\ Nb_{desc} &= Nb_{Agent} + 2.Nb_{SeD} \\ Nb_{act} &= 2.(Nb_{Agent} + 1) \end{aligned}$$

[2]JuxMem [9] (Juxtaposed Memory)est un service de partage de données sur la grille

Considérons, dans le cas d'un service par SeD, $S_{log}(req)$ la taille des messages envoyés par les éléments DIET au *LogCentral* en fonction du nombre de requêtes req. Ces messages sont composés d'une partie descriptive S_{desc} (taille des messages de type descriptif) et d'une partie $req.S_{act}$ proportionnelle au nombre de requêtes effectuées par la plate-forme. Nous définissons S_{act} comme la taille d'un message avertissant d'une activité dans la plate-forme. Il est composé de S_{ask} (taille des messages ASK_FOR_SED), de S_f (la partie fixe en taille du message SED_CHOSEN), de S_{si} (la taille du message contenant la liste triée des SeDs et les paramètres ayant permis de les ordonnancer), de $\sum_{agent}(Nb_{SeD}(agent))$ (la somme du nombre de SeDs situés dans le sous arbre de chaque agent), et enfin de S_{solve} (la taille du message avertissant de la fin du calcul). Nous obtenons alors la formule suivante :

$$\begin{aligned} S_{log}(req) &= S_{desc} + req.S_{act} \\ S_{desc} &= S_{desc}^{node}.(Nb_{Agent} + 2.Nb_{SeD}) \\ S_{act} &= Nb_{agent}.S_{ask} + Nb_{agent}.S_f + \sum_{agent}(Nb_{SeD}(agent).S_{si}) + S_{solve} \end{aligned}$$

Quelques expériences ont été réalisées afin d'évaluer le passage à l'échelle de l'implantation du *LogService* dans DIET.

Expériences autour du LogService et DIET

Protocole d'évaluation

La topologie qui maximise le nombre et la taille des messages envoyés à *LogCentral* est un arbre « râteau[3] » où le nombre d'agents est identique au nombre de SeDs. Nous répétons 5 fois la même expérience afin de pouvoir faire une moyenne des résultats.

- 10 *DIETlogTools* connectés au *LogCentral* pour recevoir les messages.
- Un client fait une série de 100 requêtes à un service DGEMM (multiplication de matrice) qui est présent sur chaque SeD.
- À chaque expérience nous faisons varier le nombre d'éléments dans la hiérarchie.
  - 1 MA, 7 LAs, 8 SeDs, soit 16 *LogComponents* connectés au *LogCentral*
  - 1 MA, 15 LAs, 16 SeDs, soit 32 *LogComponents* connectés au *LogCentral*
  - 1 MA, 31 LAs, 32 SeDs, soit 64 *LogComponents* connectés au *LogCentral*

[3]Un arbre râteau est une chaîne avec toutes les feuilles attachées au dernier nœud

Nb de nœuds	Nb_{act}	min/log	moy/log	max/log	écart type
16	1800	3e-06 s	5.33e-06 s	1.8e-05 s	1.29e-06 s
32	3400	3e-06 s	6.60e-06 s	3.1e-05 s	2.30e-06 s
64	6600	3e-06 s	8.81e-06 s	1.94e-04 s	4.75e-06 s
128	13000	4e-06 s	1.53e-05 s	5.33e-04 s	1.42e-05 s

Tableau 4.1 – Comportement de logCentral : 100 requêtes d'un client avec des plates-formes de tailles différentes.

  - 1 MA, 63 LAs, 64 SeDs, soit 128 *LogComponents* connectés au *LogCentral*
- Nous utilisons 2 sites de la grille Grid'5000 pour cette expérience. Un site est utilisé pour le déploiement de la hiérarchie DIET, et un autre est réservé pour le lancement du client et des *DIETLogTools*.

Les résultats des expériences sont présentés dans le tableau 4.1. Nous présentons, au niveau du LogCentral, les temps de traitement d'un message reçu par un *LogComponent* avant de l'envoyer aux 10 DIETLogTool.

Logiquement, le temps de traitement augmente avec le nombre d'éléments dans la hiérarchie, cependant, même dans le pire des cas, il reste inférieur à 0,5 ms, avec une moyenne de 0,1 ms. Étant placé dans la configuration qui maximise le nombre de messages et la taille des messages envoyés, on peut raisonnablement penser que le LogService n'est pas un facteur limitant au passage à l'échelle de DIET.

Comme tous les systèmes de surveillance et d'information pour les systèmes distribués [27, 97], le LogService introduit un surcoût aussi bien au niveau du réseau que sur les machines que nous surveillons. Nous nous proposons d'évaluer et de quantifier ce surcoût dans une configuration DIET fixée afin de connaître l'impact de l'activation du LogService dans DIET.

Protocole d'évaluation :

- la hiérarchie DIET est constituée d'un MA et de 10 SeDs. Chaque SeD exécute le service DGEMM ;
- un seul *DIETlogTool* connecté au *LogCentral* pour recevoir les messages ;
- toutes les 20 secondes, deux nouveaux clients commencent à soumettre des requêtes. Les clients appellent le service DGEMM pour deux matrices de petites tailles (10x10) et attendent le résultat avant de soumettre à nouveau un calcul ;
- deux séries d'expériences sont menées : une avec le LogService activé, l'autre sans le LogService.

La figure 4.7 montre les courbes moyennes sur 4 expériences avec et sans le LogService actif. Nous évaluons le débit de la hiérarchie DIET en considérant le nombre de requêtes par seconde que le MA traite.

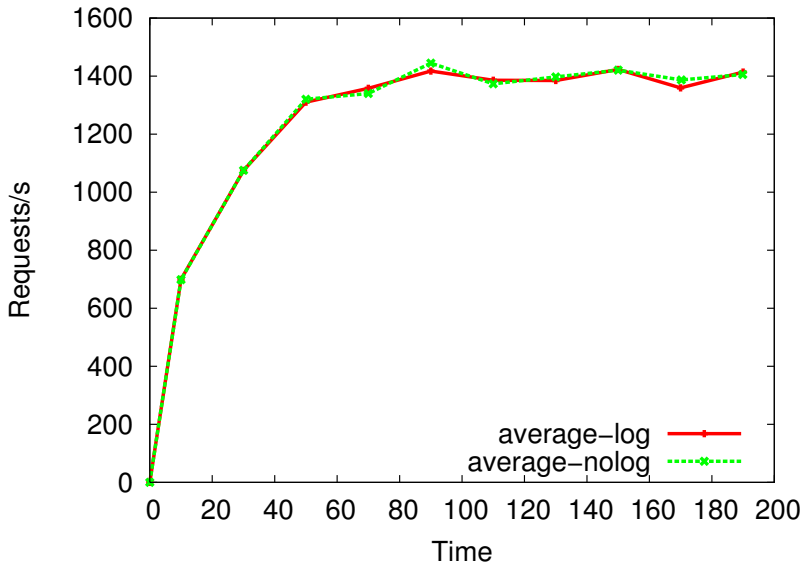


Figure 4.7 – Comparaison du débit de la plate-forme DIET avec et sans le LogService.

Le but de cette expérience n'était pas de connaître les performances limites de DIET, mais de comparer ses performances lorsque le LogService est activé ou non dans des conditions d'utilisation intense. La figure 4.7 montre que le LogService n'affecte pas les performances de DIET et que le comportement de la plate-forme est identique dans les deux cas. Il est important de noter que le nombre de requêtes lors de cette expérience était colossal : 256 838 requêtes en moyenne pour chaque expérience.

De plus grâce à la formule donnée précédemment, nous pouvons vérifier avec précision qu'aucun des 1 030 825 *logEvents* n'a été perdu au cours des expériences.

4.3 D'une grille dédiée vers une grille de volontaires

Dans cette section, nous allons décrire tout le travail préparatif qui a précédé le lancement de la première phase de calcul du projet HCMD décrit dans

le chapitre 2. Nous décrirons d'abord l'application d'amarrage moléculaire : MAXDo, puis la méthode d'évaluation du programme MAXDo sur la grille de recherche Grid'5000. Enfin nous exposerons les travaux préparatifs pour le lancement et le déroulement des calculs sur la grille World Community Grid.

4.3.1 L'application : MAXDo

Le programme MAXDo est un programme d'amarrage moléculaire développé par Sophie Sacquin-Mora et al. [81] pour l'étude des interactions moléculaires. Son but est de trouver la meilleur façon d'apparier deux protéines (p_1, p_2) pour former un complexe protéique. La qualité d'une interaction protéine-protéine est évaluée par le calcul de l'énergie d'interaction (exprimée en $kcal.mol^{-1}$), qui est la somme de deux contributions :

- E_{lj} l'énergie potentielle de Léonard-Jones [65] qui reflète les propriétés physico-chimiques des différents acides aminés de la protéine.
- E_{elec} l'énergie électrostatique qui dépend de la charge électrique des protéines.

Plus l'énergie d'interaction est négative, plus l'interaction entre les deux protéines est forte. Le programme MAXDo utilise un modèle réduit des macromolécules en interaction, associé à un champ de force simplifié développé par M. Zacharia [117]. Les deux protéines (p_1, p_2) sont observées dans un référentiel en 3 dimensions. La protéine p_1 appelée *récepteur* est fixe tandis que l'autre p_2 appelée *ligand* est mobile. Les deux molécules sont rigides et la minimisation de l'énergie d'interaction est calculée suivant les 6 degrés de liberté du ligand p_2 par rapport au référentiel de la protéine fixe $p1$.

L'Algorithme 1 utilisé dans MAXDo a été spécialement adapté de manière à rendre les calculs divisibles. L'idée est de découper l'espace d'exploration des degrés de liberté du ligand par rapport au récepteur de manière à rendre les calculs d'énergie d'interaction indépendants entre eux.

Dans le programme MAXDo, les degrés de liberté du ligand sont condensés en deux paramètres :

- le *point de séparation* (x, y, z) du ligand p_2 que l'on notera i_{sep}
- l'*orientation* (α, β, γ) du ligand p_2 que l'on notera i_{rot}

Plus formellement, l'énergie d'interaction totale E_{tot} pour un couple de protéines (p_1, p_2) est définie par :

$$E_{tot}(i_{sep}, i_{rot}, p_1, p_2)$$

Ainsi, en faisant varier les paramètres i_{sep} et i_{rot} de manière à couvrir l'espace autour de la protéine fixe p_1 on obtient la carte de la surface énergétique (figure 4.8) du récepteur pour un ligand donné. Ce qui permet de localiser

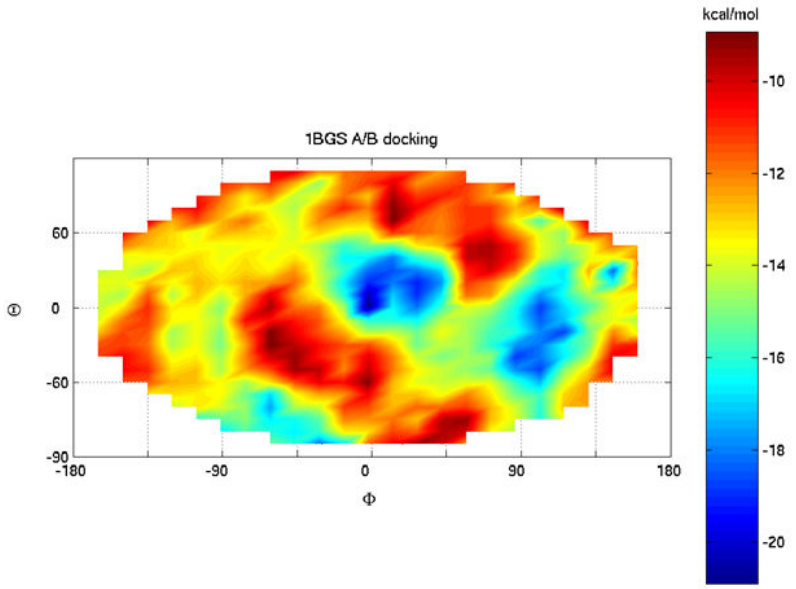


Figure 4.8 – Exemple de carte des énergies d'interaction obtenues après le *docking* entre 2 protéines.

les zones d'interaction favorables à la surface du récepteur :

$$map(p_1, p_2) = \{\forall i_{sep} \in [1..N_{sep}(p_1)], \forall i_{rot} \in [1..N_{rot}], E_{tot}(i_{sep}, i_{rot}, p_1, p_2)\}$$

Le nombre de points de séparation $N_{sep}(p_1)$ pour obtenir la carte des énergies est dépendant de la protéine fixe p_1. Ce nombre est directement lié à la forme et à la taille du récepteur p_1. Enfin si l'on cherche à obtenir les cartes d'énergies d'interactions pour un ensemble P de protéines, il faut calculer l'ensemble des énergies d'interactions défini par :

$$\forall (p_1, p_2) \in P^2, \{\forall i_{sep} \in [1..N_{sep}(p_1)], \forall i_{rot} \in [1..N_{rot}], E_{tot}(i_{sep}, i_{rot}, p_1, p_2)\}$$

En pratique, le programme MAXDo effectue deux boucles faisant varier le point de séparation i_{sep} entre deux valeurs $[N_{sep_1}, N_{sep_2}]$ et l'orientation du ligand i_{rot} entre $[N_{rot_1}, N_{rot_2}]$ afin de déterminer les énergies d'interaction entre les deux protéines (p_1, p_2) matérialisées par les zones sombres sur la figure 4.8.

Le programme MAXDo appartient à ces programmes que l'on appelle communément *embarrassingly parallel*, c'est à dire qu'il se parallélise facilement en tâches indépendantes. Ainsi, ce programme tire pleinement profit d'un environnement comportant un grand nombre de processeurs.

Données : (p_1, p_2), $(N_{sep_1}, N_{sep_2}) \in [1..N_{sep}(p_1)]^2$,
$(N_{rot_1}, N_{rot_2}) \in [1..N_{rot}]^2$
pour $i_{sep} = N_{sep_1}$ .. N_{sep_2} **faire**
  pour $i_{rot} = N_{rot_1}$ .. N_{rot_2} **faire**
    $E_{tot}(i_{sep}, i_{rot}, p_1, p_2)$
  fin
fin

Algorithme 1 : Boucles de MAXDo.

Avant de lancer le calcul sur l'ensemble des 168 protéines, une étude préliminaire [81] a été faite sur une base réduite de 5 complexes protéiques, soit 10 protéines distinctes. Cette étude a permis de démontrer la validité scientifique de cette approche. Elle a mis en évidence, en utilisant les ressources de la grille Décrypthon, les besoins considérables en temps de calcul : environ 5 ans et 1 mois de temps processeur ont été consommés pour la phase d'étude et la mise au point de MAXDo.

En considérant l'ensemble des couples (p_1, p_2) possibles pour un ensemble de protéines P, le temps de calcul croit en fonction du carré du nombre de protéines. Le passage d'un ensemble de 5 protéines à 168 protéines représente au minimum 1128 fois plus de temps de calcul.

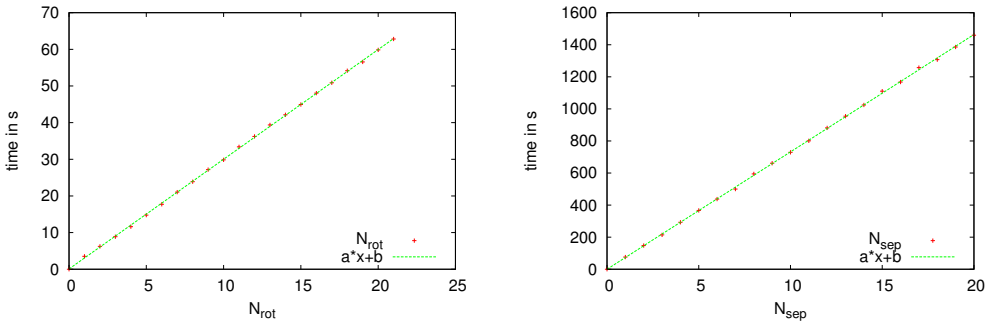
La première étape du projet HCMD consiste à obtenir les cartes énergétiques de 168 protéines issues d'une banque de données [69] de référence dont les propriétés sont connues. Les protéines de cette banque ont été choisies car elles participent toutes à au moins un complexe protéine-protéine identifié. De plus, elles couvrent un large spectre de structures et de fonctions protéiques.

Dans le but de découper les calculs à effectuer pour les 168 protéines, nous avons évalué les propriétés du programme MAXDo en fonction des paramètres d'entrée. La suite du document décrit la méthode d'évaluation ainsi que les différentes propriétés ressorties de cette évaluation.

4.3.2 Évaluation de MAXDo sur la grille Grid'5000

Avant de pouvoir lancer les calculs d'amarrage moléculaire sur les 168 protéines, nous avons évalué et vérifié quelques propriétés du programme informatique MAXDo. Comme nous l'avons décrit dans la section 4.3.1, le programme MAXDo prend en entrée quatre paramètres principaux contenus dans trois fichiers :

– deux fichiers décrivant les protéines (p_1, p_2) ;
– un fichier de paramètres contenant l'intervalle $[N_{sep_1}, N_{sep_2}]$ de point



(a) linéarité en fonction de N_{rot} ; (b) linéarité en fonction de N_{sep} ;

Figure 4.9 – Linéarités du programme MAXDo.

de séparation à calculer et l'intervalle $[N_{rot_1}, N_{rot_2}]$ d'orientation à considérer. Ce fichier contient aussi tous les autres paramètres du programme, cependant seuls les intervalles cités auparavant nous intéressent ici.

Par la suite nous noterons N_{sep} la différence $N_{sep_2} - N_{sep_1}$ et N_{rot} la différence $N_{rot_2} - N_{rot_1}$. Enfin nous appellerons $ct(N_{sep}, N_{rot}, p_1, p_2)$ le temps de calcul nécessaire pour calculer les énergies d'interaction du couple (p_1, p_2) pour l'ensemble des point N_{sep} et d'orientation N_{rot}.

L'une des premières propriétés que nous avons vérifiées est la reproductibilité des temps de calcul. Pour un jeu de paramètres donnés, le temps d'exécution est identique pour deux exécutions successives du programme sur les mêmes données. Ce temps d'exécution est principalement dépendant de la vitesse du processeur, il utilise peu de mémoire. Il ne fait d'accès disque que pour lire les fichiers d'entrée et écrire les résultats d'un calcul d'énergie. Un calcul d'énergie correspond à l'écriture d'une ligne contenant 2 entiers (index du point de séparation et de l'orientation) et 9 flottants dans un fichier.

Ensuite, nous avons vérifié que le temps de calcul pour le programme MAXDo est linéaire en fonction du nombre de points de séparation N_{sep} pour un couple (p_1, p_2) et un nombre N_{rot} d'orientation fixé. Et, respectivement, que le temps de calcul est linéaire en fonction du nombre d'orientation N_{rot} pour un couple de protéines et un nombre de points de séparation fixé. Les deux courbes présentées dans la figure 4.9 montrent cette linéarité. Pour valider la linéarité des temps de calcul nous avons fait des tests sur un ensemble de 200 couples de protéines (p_1, p_2) pris au hasard parmi les 168^2 couples possibles. Sur cet ensemble, le coefficient de corrélation linéaire était compris dans l'intervalle $[0.992, 1]$ pour chacune des deux propriétés. Les deux

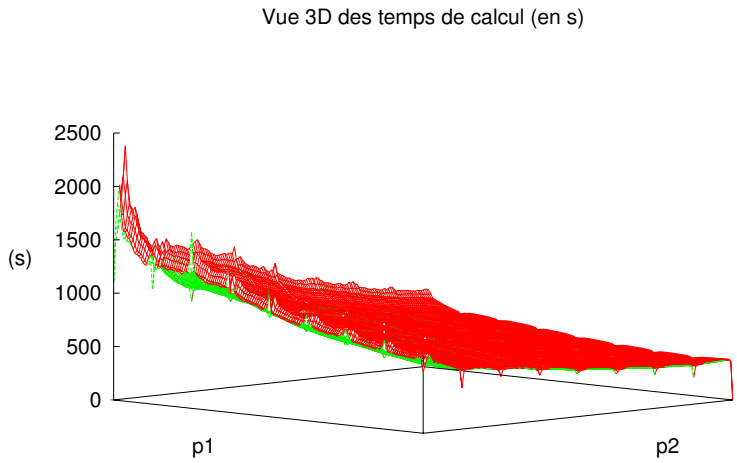


Figure 4.10 – Visualisation 3D de la matrice M_{ct} des temps de calcul pour l'ensemble des couples (p_1, p_2).

courbes de la figure 4.9 sont deux exemples parmi les 200 tests. En résumé, le programme MAXDo cumule trois propriétés :

1. **Reproductibilité**
   $\forall\ p_1, p'_1, p_2, p'_2 \in P, N_{sep}, N'_{sep}, N_{rot}, N'_{rot} \in \mathbb{N}$,
   si $p_1 = p'_1, p_2 = p'_2, N_{sep} = N'_{sep}, N_{rot} = N'_{rot}$
   alors $ct(N_{sep}, N_{rot}, p_1, p_2) = ct(N'_{sep}, N'_{rot}, p'_1, p'_2)$
2. **Linéarité** N_{sep}
   Soit p_1, p_2 et N_{rot} fixés
   alors $\forall\ N_{rot}$, $\exists\ a, b \in \mathbb{R}$,
   tel que $ct(N_{sep}, N_{rot}, p_1, p_2) = a \times ct(1, N_{rot}, p_1, p_2) + b$
3. **Linéarité** N_{rot}
   Soit p_1, p_2 et N_{sep} fixés
   alors $\forall\ N_{sep}$, $\exists\ a, b \in \mathbb{R}$,
   tel que $ct(N_{sep}, N_{rot}, p_1, p_2) = a \times ct(N_{sep}, 1, p_1, p_2) + b$

Une fois ces trois propriétés établies et vérifiées, nous avons cherché à trouver une fonction permettant de déterminer le temps de calcul en fonction des paramètres du programme. Malheureusement, nous n'avons pas pu mettre en évidence une telle fonction.

Cependant étant donné la linéarité en fonction de N_{sep} et N_{rot} pour connaître les temps de calcul pour chaque couple de protéines, il suffit de

moyenne	671 s	11 min 11 s
écart type	968,04 s	17 min 23 s
médiane	384 s	6 min 24 s
min	6 s	6 s
max	46347 s	12 h 52 min 33 s
somme	18943323 s	219 j 6 h 2 min 3 s

Tableau 4.2 – Statistique des temps de calcul avec N_{rot}=21 et N_{sep}=1 pour l'ensemble des 168^2 couple de protéines possible.

lancer le programme MAXDo sur chaque couple de protéines possible pour un nombre restreint de N_{sep} et N_{rot}. Ainsi nous obtenons une matrice nommée M_{ct} qui définit le temps de calcul pour un couple de protéine (p_1, p_2) et pour un nombre de points de séparation et d'orientations fixés. Les deux propriétés de linéarité nous permettent d'obtenir les temps de calcul réels par simple multiplication. La courbe de la figure 4.10 illustre, sur un graphique en trois dimensions, les temps de calcul contenus dans la matrice M_{ct}. Nous avons établi ces temps de calcul en fixant le nombre N_{rot}=7 et N_{sep}=1. Une seule propriété est mise en évidence : plus la taille des 2 protéines est importante, plus les temps de calcul sont élevés, mais il y a des exceptions.

Le tableau 4.2 présente les statistiques sur les valeurs contenues dans la matrice. L'importance de l'écart-type et l'étendue des valeurs confirment la dispersion élevée des temps de calcul. Cependant 98 % des temps de calcul pour N_{rot}=21 et N_{sep}=1 sont inférieurs à 1 heure. Il est aussi important de noter la quantité totale de temps processeur qu'il a été nécessaire pour obtenir cette matrice M_{ct} : 73 j 2 h 41 s. Nous avons utilisé pour cela 640 processeurs identiques (Opteron 246 @ 2 GHz) de la grille de recherche Grid'5000, ce qui nous a permis d'obtenir l'ensemble de ces résultats en à peine une demi-journée.

Enfin le nombre de points de séparation N_{sep} pour chaque protéine est déterminé par un programme à part qui prend en entrée une protéine. Le tableau 4.2 présente les statistiques sur les valeurs de N_{sep} nécessaires pour l'amarrage des 168 protéines. La figure 4.11 présente l'histogramme des effectifs des protéines ayant un nombre N_{sep} compris entre 2 valeurs. 92% des protéines ont moins de 3162 points de séparation.

Le nombre d'orientations N_{rot} est un paramètre qui a été déterminé lors de l'étude préliminaire [81]. Les scientifiques ont fixé cette valeur à N_{rot}=21, ce qui correspond en réalité à 210 variations pour les trois degrés de liberté (α, β, γ) du ligand, la protéine mobile.

moyenne	écart type	médiane	min	max	somme
1753.17	1214.04	1430	301	9840	294533

Tableau 4.3 – Statistique des points de séparation.

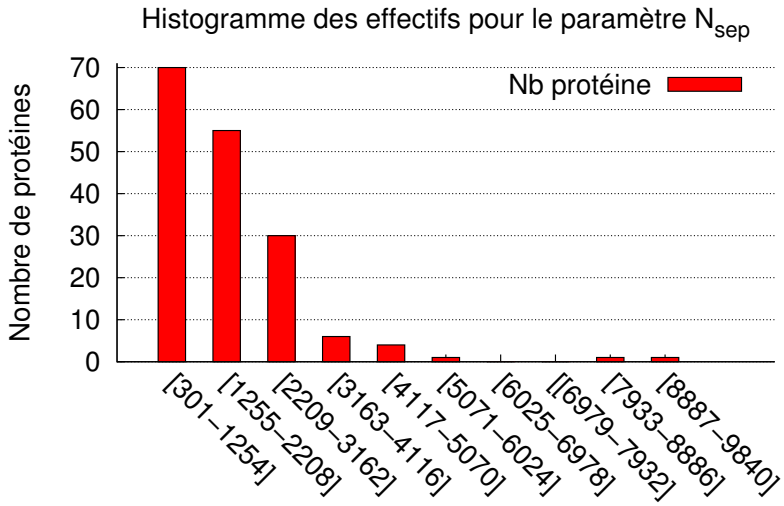


Figure 4.11 – Histogramme des effectifs des protéines pour le paramètres N_{sep}.

Avec l'ensemble de ces données : la matrice des temps de calcul, le tableau donnant le nombre de points de séparation $N_{sep}(p)$ pour une protéine p et le paramètre N_{rot} fixé à 21, nous sommes en mesure d'estimer le temps processeur nécessaire à l'amarrage moléculaire croisé des 168 protéines sur notre processeur de référence (Opteron 246 @ 2GHz). Cette quantité de calcul est donnée par la formule :

$$CPU_{total}^{G5K} = \sum_{p_1,p_2 \in P} N_{\text{sep}}(p_1) \times N_{rot} \times ct_{\text{iter}}(p_1, p_2) \tag{4.1}$$

CPU_{total}^{G5K} représente un peu plus de 1489 années de temps[4] de calcul sur notre processeur de référence. En d'autres termes si nous espérions finir les calculs en 2007 sur notre processeur de référence, il aurait fallu lancer les calculs à l'époque où Thierry 1^{er}, successeur de Clovis, était roi de France.

[4]exactement 14 siècles 88 ans 237 jours 19 heures 45 minutes 54 secondes

4.3.3 Préparation pour la grille d'internautes

Une fois la matrice des temps de calcul obtenue, il nous reste à découper les calculs en paquets de tâches ou unités de travail : appelés *workunits*. D'après la description que nous avons faite de l'application MAXDo, nous pouvons jouer sur deux paramètres : le nombre de points de séparation N_{sep} et le nombre d'orientations N_{rot}. Les besoins de la grille de volontaires World Community Grid sont les suivants :

- la quantité de données envoyées aux internautes doit être faible (de l'ordre de quelques Mo) ;
- la durée d'exécution totale d'une *workunit* doit être d'environ 10 heures ;
- la quantité de données résultats d'une *workunit* doit être aussi de l'ordre de quelques Mo.

Nous avons vu que le programme MAXDo nécessite trois fichiers : deux fichiers au format pdb [13] qui décrivent les structures 3D des protéines et un fichier qui dépeint les paramètres du programme (N_{sep_1},N_{sep_2},N_{rot_1},N_{rot_1}, *etc*). Le fichier contenant les résultats dépendent du nombre de calculs effectués. Il contient une ligne pour un calcul d'énergie. C'est à dire pour une *workunit* contenant $N_{sep} = N_{sep_2} - N_{sep_2}$ points de séparation et $N_{rot} = N_{rot_2}$-N_{rot_1} orientations, le fichier résultat contiendra $N_{sep} \times N_{rot}$ lignes de résultats soit 132 octets pour chaque ligne.

Compte tenu de la distribution des temps de calcul (figure 4.10 et tableau 4.2), il est inutile de découper le nombre d'orientations fait lors d'un calcul sauf éventuellement pour quelques protéines que nous pouvons traiter à part. Il suffit donc de déterminer, pour un couple de protéines, le nombre de points de séparation à calculer, le nombre d'orientations étant fixé à 21. Il n'y a pas de difficultés quant à la taille des données. En effet, les quantités de données en entrée sont toujours faibles et les résultats produits sont directement liés au nombre de N_{sep} calculés. Pour donner un ordre de grandeur, le calcul des énergies d'interaction, pour 2000 points de séparation en considérant 21 orientations différentes, donne un fichier de résultats d'une taille de 5 Mo 294 Ko.

Ainsi le problème de préparation des *workunits* devient le suivant :

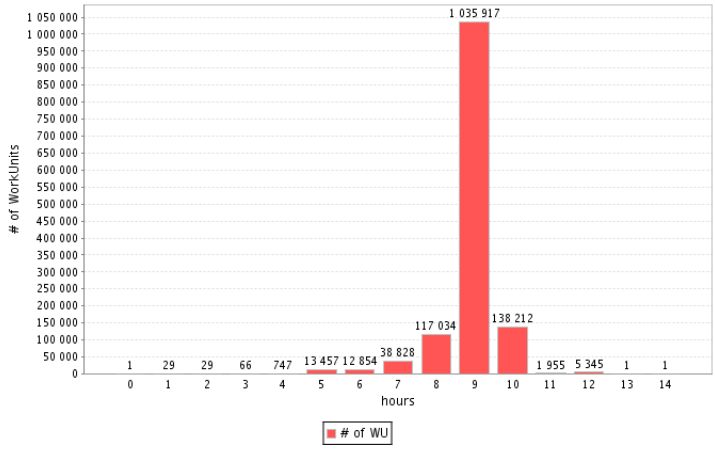
Soit h la durée voulue d'une *workunit*, notons $ct(p_1, p_2)$ la durée du calcul pour effectuer 21 orientations et 1 point de séparation :

$$\begin{array}{lll} \forall (p_1,p_2) \in P, \text{ trouver} & n_{\text{sep}} \in [1..N_{\text{sep}}(p_1)] \text{ tel que} & \\ \text{si} & \left\lfloor \frac{h}{\text{ct}(p_1,p_2)} \right\rfloor \leq 1, & n_{\text{sep}} = 1 \\ \text{si} & \left\lfloor \frac{h}{\text{ct}(p_1,p_2)} \right\rfloor \geq N_{\text{sep}}(p_1), & n_{\text{sep}} = N_{\text{sep}}(p_1) \\ & \text{sinon} & n_{\text{sep}} = \left\lfloor \dfrac{h}{\text{ct}(p_1,p_2)} \right\rfloor \end{array}$$

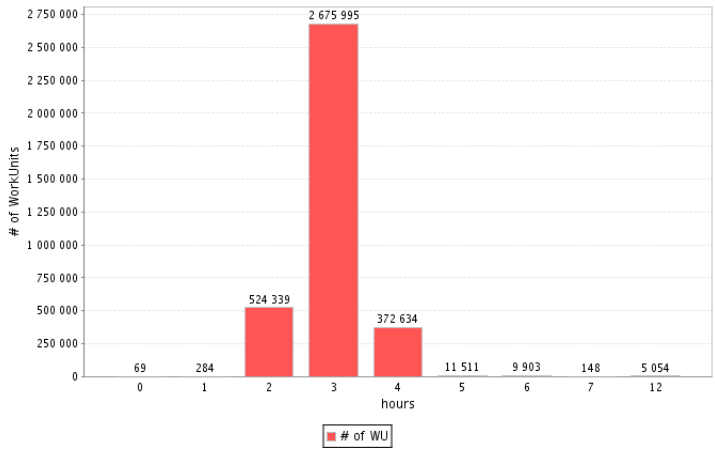
Ce paramètre h est imposé par l'équipe du World Community Grid. Elle désire avoir un temps de calcul pour chaque *workunit* d'environ 10 heures. À titre de comparaison, dans l'étude menée par Kondo et al. [57] sur des grilles volatiles d'entreprises, les auteurs trouvent que le temps moyen d'exécution disponible sur une machine est de 2,6 heures. Plusieurs justifications expliquent le choix empirique de fixer h à 10 heures :

- la première explication technique vient des capacités des serveurs qui distribuent les *workunits* et reçoivent les résultats. En allongeant les temps de calcul envoyés aux internautes, la charge des serveurs est diminuée. Une étude [8] montre qu'un serveur BOINC est capable de servir un peu moins de 9 millions de tâches par jour. Les serveurs du World Community Grid seraient tout à fait capables de supporter une charge plus importante, mais cela permet d'assurer une certaine fiabilité au système et une marge confortable pour la montée en charge. Ainsi, le nombre total de *workunit* est diminué.
- La deuxième explication est plus psychologique, ou humaine. Le fait de fixer la durée d'un *workunit* à 10 heures correspond à une durée suffisante pour qu'un internaute se rende compte que son ordinateur travaille. Une durée trop faible donnerait l'impression subjective que le travail donné n'est pas très compliqué, et donc peu important.
- Enfin cette valeur, permet aussi d'avoir une progression quasi quotidienne du travail effectué. Ainsi, les statistiques du site internet du World Community Grid sont rafraîchies tous les jours.

En tenant compte de la valeur de ce paramètre h les figures 4.12(a) et 4.12(b) montrent deux exemples de générations de *workunits*. Il existe beaucoup de façons de découper le travail en paquets de tâches. Nous avons utilisé une heuristique qui permet de réduire le nombre total de *workunits* en s'autorisant à dépasser la valeur h donnée en entrée. Au final, c'est l'équipe du World Community Grid qui a été seule maître de ce découpage.



(a) $h = 10$ heures, Nb wu = 1 364 476 ;



(b) $h = 4$ heures, Nb wu = 3 599 937 ;

Figure 4.12 – Deux exemples de distributions des temps de calcul de workunits.

Avant de pouvoir lancer les calculs sur la grille d'internautes, une dernière modification a été apportée au programme MAXDo. Nous avons introduit un mécanisme de reprise sur arrêt, permettant de reprendre le calcul au plus proche d'un point de sauvegarde. Concrètement, nous avons ajouté la possibilité à MAXDo de pouvoir reprendre le calcul au dernier point de séparation calculé. Si le programme est coupé au milieu des calculs d'énergies pour un point de séparation du ligand, le calcul reprendra à ce point de séparation. Cependant les calculs déjà effectués (variation d'orientations) pour ce point de séparation seront perdus : nous ne sauvons les calculs qu'au bout des 21 orientations.

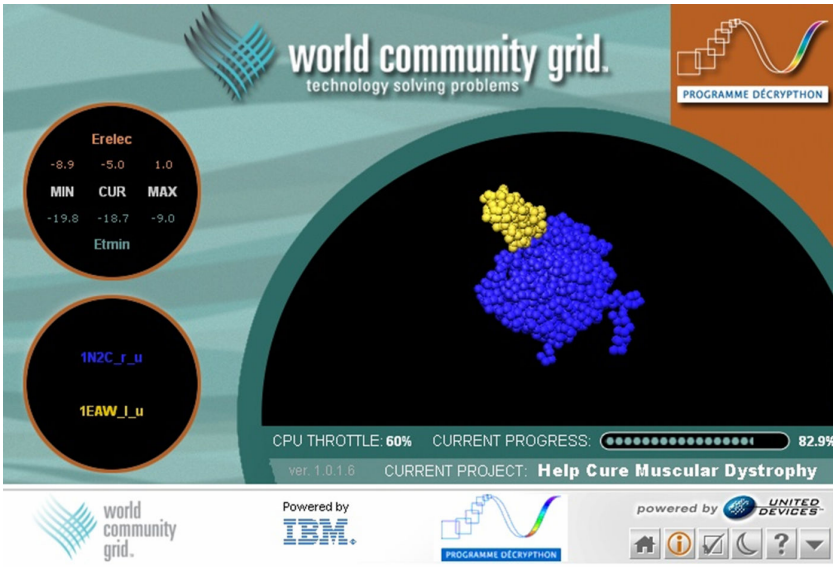


Figure 4.13 – Économiseur d'écran du programme MAXDo.

Enfin, l'équipe du World Community Grid a ajouté une interface graphique (figure 4.13) au programme MAXDo. Celle-ci montre les deux protéines qui sont en cours d'analyse. Elle affiche, de plus, la valeur des énergies calculées et la progression du calcul. Cette interface peut être affichée en tant qu'économiseur d'écran ou tout simplement lorsque le programme est actif.

À ce stade, tout le travail préparatoire est effectué. Nous sommes prêts à lancer le projet sur la grille de volontaires du World Community Grid.

4.3.4 Déroulement des calculs sur la grille WCG

La phase de calcul pour le projet HCMD sur la grille de volontaires du World Community Grid a commencé le 19 Décembre 2006. Elle s'est terminée le 11 juin 2007. L'ensemble des calculs d'amarrage moléculaire croisés entre les 168 protéines a donc duré 26 semaines. Comme prévu, l'ensemble des résultats constitue un peu moins de 128 Go de données, soit plus de 1 milliard 39 millions calculs d'énergie d'interaction[5].

Vérification des résultats

Durant la phase de calcul, les résultats étaient envoyés par l'équipe du World Community Grid sur un serveur de données du projet Décrypthon.

[5]exactement 1 039 112 424 = 294 533 x 168 x 21 d'énergie d'interaction calculées

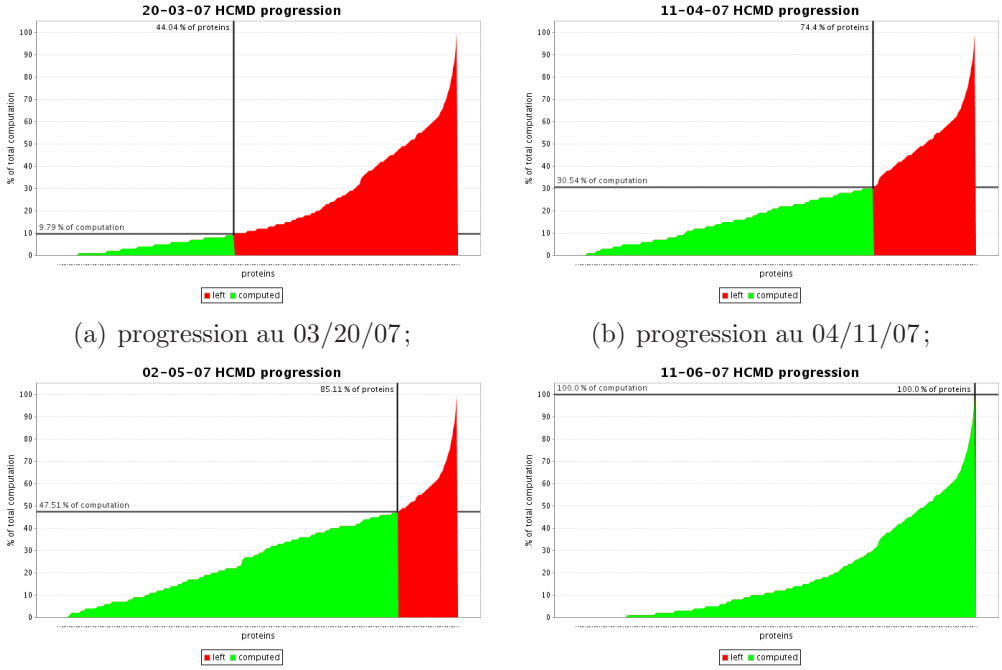
Nous validions les fichiers reçus chaque semaine afin de vérifier plusieurs éléments :

- le nombre de résultats présents. Cette vérification est basée sur le nombre de lignes contenues dans chaque fichier résultat. Un fichier correspond à un nombre variable de N_{sep}. Nous vérifions alors que le nombre de lignes d'un fichier résultat d'une *workunit* contenait exactement $N_{sep} \times N_{rot}$, N_{sep} et N_{rot} étant les paramètres du *workunit*.
- La valeur des résultats de chaque ligne. Cette vérification est possible. En effet, nous savons que les valeurs contenues dans les fichiers résultats ont un intervalle de validité. Ainsi 2 des 11 valeurs sont les index représentant les points de séparation et l'orientation du ligand par rapport au récepteur. 3 des 9 autres correspondent aux angles (α, β, γ) d'orientation du ligand. Les valeurs sont nécessairement comprises dans l'intervalle $[-2\pi, 2\pi]$. Enfin 3 des 6 dernières valeurs représentent les énergies de minimisation qui doivent être négatives. Lorsque ces valeurs sont nulles, le calcul de l'énergie d'interaction pour les paramètres donnés est alors un échec.

Ces vérifications ont donné lieu à une méthode de vérification des fichiers retournés par les internautes. Par défaut, une des méthodes [7] employée par les grilles de volontaires, est la validation des résultats en effectuant des calculs redondants sur la même *workunit*. Tant que l'on n'obtient pas un ensemble de résultats identiques (*quorum*) sur les mêmes jeux de données, le calcul est refait. Cette méthode est très sûre, en revanche, elle gaspille beaucoup de ressources, au moins 2 fois plus que nécessaire. Elle vise principalement à rejeter les résultats qui ont pu être altérés par un volontaire, ou endommagés lors d'un transfert ou d'une manipulation quelconque.

La validation du contenu des résultats permet d'envoyer une seule copie d'une *workunit*, d'attendre le résultat, de le vérifier, et de le renvoyer s'il ne correspond pas aux critères de validité. Toutefois, après avoir mis cette méthode de validation en place, il existait toujours des calculs redondants. Pour chaque *workunit* est défini un temps d'attente maximum (*timeout*). Au bout de ce temps, une nouvelle copie du même *workunit* est envoyée à un autre volontaire. Néanmoins, le résultat de la première copie peut quand même être retourné après ce *timeout*, ainsi un résultat supplémentaire sera comptabilisé.

Sur l'ensemble des calculs effectués sur la grille World Community Grid le facteur de réplication a été de 1,37. Ce qui veut dire que 73% des résultats produits par la grille d'internautes étaient utiles au projet. Nous obtenons ce facteur en effectuant le rapport entre le nombre de résultats reçus par les serveurs du World Community Grid (5 418 019) et le nombre de résultats utiles (3 936 009) que nous avons reçus. Ce facteur est plutôt bon compa-



(a) progression au 03/20/07 ; (b) progression au 04/11/07 ;

(c) progression au 05/02/07 ; (d) progression au 06/11/07 ;

Figure 4.14 – La progression du projet HCMD.

rativement au facteur nécessairement supérieur à 2 obtenu avec la méthode classique [7] de vérification des résultats.

Progression des calculs

Chaque semaine de la phase de calcul, nous avions une réunion téléphonique entre l'équipe du World Community Grid et les équipes du programme Décrypthon pour nous assurer du bon déroulement du projet et des calculs. Lors de cette réunion l'équipe responsable de la grille d'internautes nous fournissait les temps de calcul et le nombre total de résultats générés par les internautes. À partir de ces données, nous avons réalisé un graphique exhibant l'avancement du projet. C'est aussi à partir de ces informations que nous avons déduit le facteur de réplication des calculs effectués par les volontaires.

La figure 4.14 contient quelques exemples de courbes de progression du projet HCMD. En abscisse se trouve le pourcentage de protéines calculées, en ordonnée nous représentons le pourcentage cumulé des calculs effectués. Nous avons construit ces courbes en faisant correspondre les temps de calcul de la matrice M_{ct} et les résultats envoyés par l'équipe du World Community Grid.

Cette progression était aussi affichée sur un site internet [6] afin de permettre aux volontaires de suivre le déroulement des calculs.

Nous retrouvons sur ces courbes le caractère hétérogène des temps de calcul nécessaires pour l'amarrage moléculaire d'une protéine avec toutes les autres. Nous remarquons, de plus, que l'équipe du World Community Grid a choisi de débuter par les protéines demandant le moins de calcul. Ce choix a permis de vérifier rapidement les calculs qui ne coûtaient pas trop « cher », en outre, les calculs redondants issus de la méthode classique de vérification n'ont pas été faits sur des protéines trop coûteuses (en temps de calcul). En contre partie, même si, au début, nous observions une progression assez rapide en nombre de protéines calculées, à la fin, la progression était beaucoup plus lente. Ainsi au 2 mai 2007 il restait à peine 15 % des protéines à calculer, ce qui représentait pourtant encore 50 % du temps processeur total nécessaire.

4.3.5 Analyse des résultats de la phase I du projet HCMD

À l'issue de la phase de calcul du projet HCMD sur le World Community Grid, plus de 128 Go de données ont été générées. L'ensemble de ces données représente les calculs d'énergies d'interaction entre 2 couples de protéines parmi les 168 protéines de la base.

Ces données ont été calculées par les internautes. Le premier travail consistait à recombiner les fichiers afin de masquer le découpage. En effet, nous l'avons décrit précédemment, celui-ci avait été nécessaire pour donner des *workunits* de 10 heures aux internautes. Nous sommes donc passés de 3 936 009 à 28224 fichiers résultats. Chaque fichier contenant $N_{sep}(p_1) \times N_{rot}$ lignes. Parmi les résultats obtenus, seule l'énergie minimum, pour l'ensemble des 21 orientations de la protéine, est intéressante pour les scientifiques. Les fichiers ont donc encore été manipulés dans le but de filtrer uniquement les énergies minimum pour un point de séparation donné. Ce filtrage à permis de diviser par 21 le nombre de lignes dans chaque fichier.

Nous avons donc désormais un ensemble de fichiers pour les 168 protéines contenant $N_{sep}(p_1)$ lignes chacun. Les résultats sont présents sous la forme de 168 dossiers (archives), un dossier par protéine contenant l'ensemble des 168 calculs d'énergie d'interaction avec les 168 autres protéines.

Cette simple manipulation de fichiers et de filtrage du contenu a été effectuée en même temps que la vérification des données que nous effectuions lorsque les résultats nous étaient retournés par l'équipe du World Community

[6] `http://graal.ens-lyon.fr/~rbolze/hcmd.html`

Grid. Au final, la quantité de données utile pour les scientifiques représente 6 Go 59 Mo, répartie sur un total de 168^2 (28224) fichiers.

Une fois cette étape importante réalisée, grâce à la grille World Community Grid, il reste à faire l'analyse des résultats. Cette étape, bien que moins gourmande en temps de calcul, consiste à produire les cartes d'énergie d'interaction, à récolter des données statistiques sur les énergies et à déterminer les résidus situés aux interfaces protéiques pour chaque protéine. Ces calculs ont besoin de toutes les données issues de la minimisation d'énergie d'interaction pour un couple de protéines (p_1, p_2). De plus, ces analyses génèrent une grande quantité de données, incompatibles avec les contraintes d'une grille de volontaires. Nous ne pouvions donc pas l'effectuer sur les machines des internautes.

Nous avons donc réalisé cette étape d'analyse pour les 168 protéines, sur la grille dédiée Décrypthon, plus adaptée à des calculs longs qui manipulent une quantité importante de données. Au final pendant un mois de calcul sur la grille dédiée Décrypthon, l'analyse a généré plus de 406 Go de données non compressées et a consommé près d'un an et demi [7] de temps processeur. Sur les 168 analyses, la durée moyenne d'analyse était de 2 jours et 21 heures, avec un maximum de presque 25 jours et un minimum à un peu moins de 7 heures, l'écart type étant de 3,3 jours.

4.4 Comparaison de deux types de grilles

Le site web du World Community Grid indique qu'un total de plus de 80 siècles[8] de temps processeur a été consommé par l'ensemble des calculs du projet HCMD. Cette quantité notée CPU_{total}^{WCG} est 5,43 fois plus grande que la valeur CPU_{total}^{G5K} que nous avons estimée sur le processeur de référence (Opteron 246 @ 2 GHz) de la plate-forme Grid'5000. Ce total comptabilise également les calculs redondants. Si nous prenons en compte le facteur de réplication ($rep_{fact} = 1,37$), le rapport devient :

$$\frac{CPU_{total}^{WCG}}{CPU_{total}^{G5K} \times rep_{fact}} = 3.96$$

Le rapport ainsi trouvé peut être confirmé en examinant le découpage qui a été fait pour les calculs envoyés aux internautes. Comme nous l'avons écrit dans la section 4.3.3, c'est l'équipe du World Community Grid qui a décidé du découpage des *workunits* en se basant sur les indications que nous

[7] exactement 1 a 121 j 18 h 16 min 36 s

[8] exactement 80 s 82 a 275 j 17 h 15 min 44 s

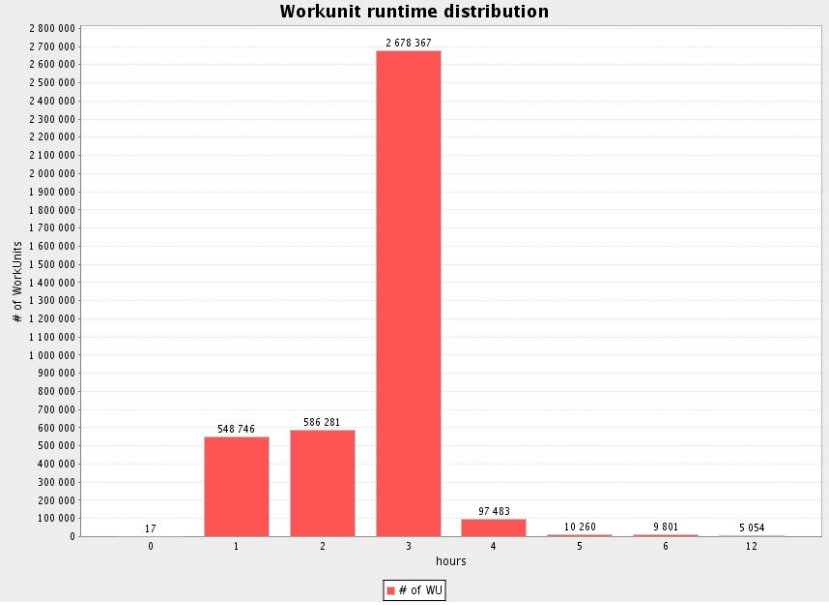


Figure 4.15 – Distribution des temps de calcul des *workunits* ayant effectivement été exécutés sur les machines des volontaires.

leur avions données et sur la matrice M_{ct}. À partir des fichiers résultats reçus nous avons reconstruit ce découpage. La figure 4.15 montre la distribution des temps de calcul des *workunits* en utilisant la matrice M_{ct}. Le nombre total de *workunits* est de 3 936 009. Le temps d'exécution moyen (toujours en utilisant la matrice M_{ct}) est de 3 h 18 min 47 s (soit 11927 s).

En prenant la somme des temps processeurs des internautes et le nombre total de résultats produits, la moyenne des temps processeurs par résultat est de 13 h 4 min 6 s. Le rapport entre ces deux moyennes est ($\frac{47046}{11927}$).

$$\frac{\overline{CPU}^{WCG}}{\overline{CPU}^{G5K}} = 3.94$$

Ces deux méthodes de calcul corroborent le fait qu'il existe un facteur 4 entre les temps processeurs nécessaires sur la grille World Community Grid et le temps processeur sur notre machine de référence Opteron 246 @ 2GHz. Nous appellons ce rapport : le ralentissement ou *slowdown*.

Explication du *slowdown*

Plusieurs arguments expliquent cette différence de performance entre la grille World Community Grid et le temps estimé sur le processeur de référence de Grid'5000.

- Premièrement, l'agent UD World Community Grid compte le temps total actif du programme MAXDo. Ce temps est appelé le *wall clock time*. Il ne représente pas exactement le temps que le processeur passe sur l'application, mais le temps où l'application est active.
- Deuxièmement, l'agent UD World Community Grid est configuré pour n'utiliser que 60% du temps processeur. Cette configuration ne peut être changée que si l'utilisateur télécharge un utilitaire spécial qui changera la valeur par défaut.

Ces deux faits à eux seuls peuvent expliquer au moins 50% du facteur 4 de ralentissement. Les 50% restant peuvent être expliqués par le caractère volatil et non dédié des machines du World Community Grid. En effet, le volontaire peut à tout moment décider d'arrêter sa machine ou tout simplement couper l'agent du World Community Grid. Dans ce cas, l'application scientifique est stoppée sans préavis, et elle sera relancée par l'agent à son dernier point de sauvegarde (*checkpoint*).

Enfin, les machines des volontaires du World Community Grid sont en moyenne moins rapides que le processeur de référence que nous avons utilisé pour l'évaluation des temps de calcul. L'équipe du World Community Grid a estimé que le processeur moyen de la grille de volontaires pouvait être comparé à un processeur @ 1,8 GHz au moment où le projet HCMD s'est exécuté sur la grille. Il faut aussi ajouter que le binaire de l'application MAXDo n'a pas été compilé avec les mêmes options d'optimisation agressives, entre la version qui s'est exécutée sur Grid'5000 et la version sur le World Community Grid. Nous ne pouvions utiliser ces mêmes options car certaines d'entre elles tirent partie des instructions spécifiques des processeurs. Or le binaire utilisé dans l'agent UD World Community Grid est spécifique au système, mais pas au type de processeur. Il est donc indispensable d'avoir un binaire compatible avec le plus grand nombre de processeurs.

Toutes ces raisons expliquent le facteur 4 d'augmentation du temps processeur observé entre l'estimation sur le processeur de référence Opteron 246 @ 2 GHz et le temps observé sur la grille World Community Grid. Cependant ce rapport est largement compensé par la quantité de processeurs disponibles, qui s'inscrit sur des durées beaucoup plus longues. Il est, en l'état, impossible à cause de règles évidentes de partage, de mobiliser pendant plus de 2 jours tout Grid'5000, d'autant plus que cette plate-forme n'est pas une grille de production.

4.4.1 Processeurs virtuels à plein temps

La grille World Community Grid est une grille de volontaires qui s'enregistrent sur le site internet du même nom. Actuellement la plate-forme

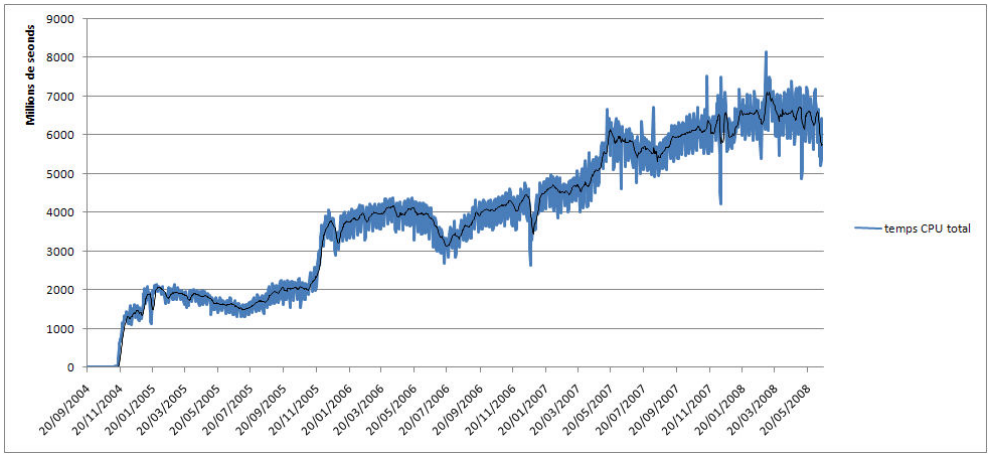


Figure 4.16 – Temps total d'exécution quotidien depuis le début du World Community Grid.

compte près de 400 000 membres et plus d'un million de machines. Le site internet du World Community Grid fournit des statistiques quotidiennes sur la durée d'exécution totale, le nombre de points générés et le nombre de résultats renvoyés chaque jour. À partir de ces données, la courbe de la figure 4.16 montre les durées d'exécution totales quotidiennes du World Community Grid.

Afin de pouvoir comparer la grille World Community Grid avec d'autres grilles, nous avons décidé d'introduire la notion de *processeur virtuel équivalent plein temps*.

Nous définissons un *processeur virtuel plein temps*, noté P_{vfp}, comme la somme des temps processeurs générés sur une période ramenée à la durée de cette période. Cette définition répond à la question suivante :

> *Combien faut-il de processeurs pour générer* $\sum t_{CPU}$ *temps processeur sur une durée* T *?*

Nous appelons cette quantité *processeur virtuel équivalent plein temps* car elle ne représente pas un processeur réel mais le nombre de processeurs minimum qu'il est nécessaire d'avoir pour générer $\sum t_{CPU}$ pendant une durée T. Ces processeurs ne sont pas forcément identiques, il n'ont pas la même puissance et ils n'ont pas nécessairement travaillé pendant toute le durée T. Nous connaissons uniquement le temps consacré à la grille World Community Grid au cours de la période T. Ainsi, par exemple, si un siècle de temps d'exécution a été généré durant 1 jour, cela veut dire qu'il a fallu au moins 36 500 P_{vfp} pour le produire.

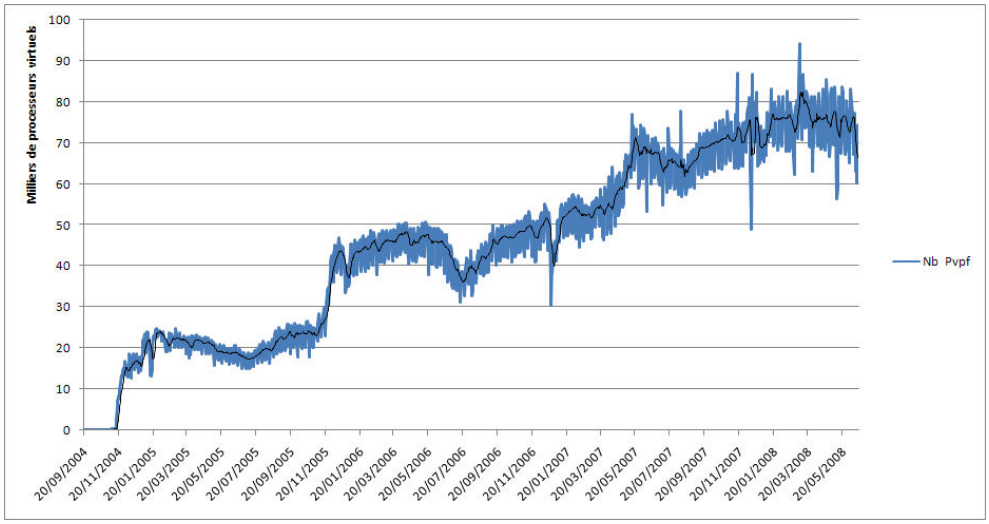


Figure 4.17 – Nombre de *processeurs virtuels équivalent plein temps* depuis le début du World Community Grid.

À partir de la définition de *processeur virtuel équivalent plein temps*, la figure 4.17 donne le nombre de P_{vfp} qui travaillent chaque jour pour la grille World Community Grid.

Le nombre de *processeurs virtuels équivalent plein temps* est en augmentation depuis le début de la grille World Community Grid. Au cours de cette année 2008 le nombre moyen est de 75 239, avec un écart type de 5 963, un minimum de 56 463 et un maximum de 94 287.

Concentrons nous maintenant sur la période entre le 19 Décembre 2006 et le 11 Juin 2007 pendant laquelle les calculs du projet HCMD ont été effectués. Au cours de la phase de calcul, nous avions un rapport hebdomadaire sur la quantité de temps processeur consacré au projet HCMD. La figure 4.18 montre les *processeurs virtuels équivalent plein temps* dédiés au projet HCMD (courbe P_{vfp} HCMD) et les P_{vfp} totaux du World Community Grid. Une troisième courbe (P_{vfp} HCMD utile) trace uniquement les P_{vfp} ayant été utiles au projet HCMD, c'est à dire qui n'ont pas fait un calcul redondant.

5 périodes peuvent être distinguées :

A - quasiment aucun P_{vfp} ne participe au projet HCMD. Durant cette période, la priorité par rapport aux autres projets était très faible, moins de 1 % des calculs réalisés étaient consacrés au projet HCMD ;

B - la priorité des calculs du projet HCMD change progressivement, la proportion des P_{vfp} attribuée au projet HCMD augmente ;

C - la priorité du projet HCMD reste inchangée, le nombre de P_{vfp} suit

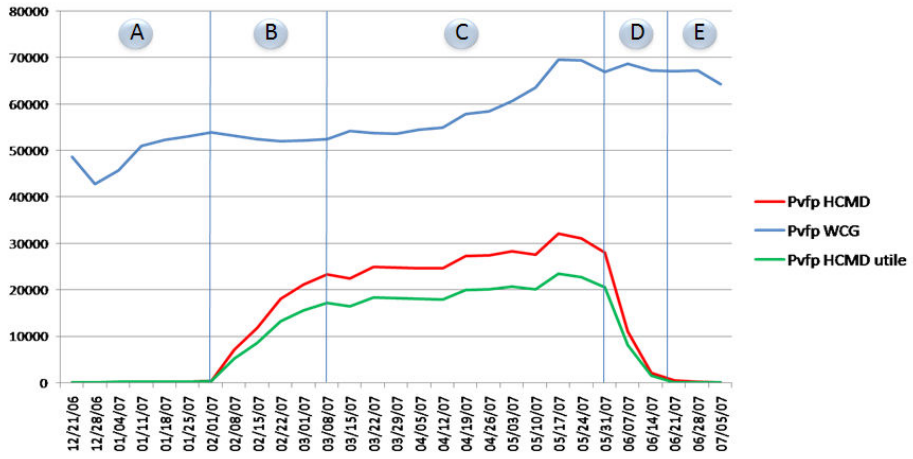


Figure 4.18 – Nombre de *processeurs virtuels équivalent plein temps* durant la phase active du projet HCMD.

l'évolution générale de la plate-forme World Community Grid. Durant cette période 45 % des calculs effectués par la plate-forme étaient destinés au projet HCMD ;

D - L'ensemble des *workunits* du projet HCMD a été dispatché aux volontaires, la proportion des P_{vfp} diminue ;

E - L'ensemble des résultats utiles a été reçu, cependant il y a encore quelques résultats qui sont retournés. Ces résultats sont comptabilisés car il faut attribuer des points aux volontaires qui ont effectué ces calculs, même si nous savons qu'ils sont inutiles.

Pendant la période du 19 Décembre 2006 au 5 juillet 2007 (A+B+C+D+E), le nombre moyen de P_{vfp} de la plate-forme World Community Grid entière était de 57 551, l'écart type 8 662, le minimum de 30 343 et le maximum de 76 983. Durant cette même période, le nombre moyen de P_{vfp} consacré au projet HCMD était de 15 006, l'écart type de 12 405, le minimum de 52 et le maximum de 32 136.

Sur la période C (du 08-03-2007 au 31-05-2007 environ 3 mois), que l'on qualifiera de période de pleine activité, le nombre moyen de P_{vfp} de la plate-forme World Community Grid entière était de 59 911, l'écart type de 7 322, le minimum de 46 421 et le maximum de 76 983. Durant cette même période le nombre moyen de P_{vfp} consacré au projet HCMD était de 23 504, l'écart type 7 415, le minimum 468 et le maximum 32 136.

Sans les deux longues périodes A+B quasi inactives (un peu plus de 2

Type de grille	Total (A+B+C+D+E)	Pleine activité (C)
World Community Grid	15 006	23 504
Grille dédiée	2 764	4 328

Tableau 4.4 – Tableau d'équivalence entre le P_{vfp} de World Community Grid et le processeur de référence de la grille dédiée Grid'5000 pour le projet HCMD.

Type de grille	année 2008
World Community Grid	75 240
Grille dédiée	13 856

Tableau 4.5 – Tableau d'équivalence entre le P_{vfp} moyen de World Community Grid et le processeur de référence de la grille dédiée Grid'5000 pour l'année 2008.

mois), la phase de calcul du projet HCMD aurait duré environ 3 mois. En effet, nous pouvons estimer que l'ensemble des calculs faits pendant la période A+B auraient pu être fait en une semaine dans la situation de la phase C.

Le tableau 4.4 met en relation le nombre de *processeurs virtuels équivalent plein temps* de World Community Grid et le processeur de référence de la grille Grid'5000 grâce au coefficient 5,43. Ici, nous prenons en compte le facteur de ralentissement (≈ 4) et le coefficient de réplication (1,37), car cette réplication est une caractéristique du mode de fonctionnement de la grille de volontaires. La grille World Community Grid a donc permis de jouer un rôle équivalent à la grille Grid'5000 sur une période consécutive de 6 mois (début 2007, Grid'5000 comptait environ 2500 processeurs). Et si l'on se restreint à la période de pleine activité, il aurait fallu disposer d'une plate-forme 1,6 fois plus importante que Grid'5000 sur une durée de 3 mois pour pouvoir effectuer les mêmes calculs.

En étudiant le nombre moyen de *processeurs virtuels équivalent plein temps* disponibles en permanence dans la plate-forme World Community Grid pour l'année 2008 ; le tableau 4.5 donne l'équivalence en appliquant le facteur 5,43.

Plus précisément, sur l'année 2008, l'écart type est de 5 963, la valeur maximale est de 94 287 et la valeur minimale est de 56 463.

Il est important de garder à l'esprit que le facteur 5,43 n'est vrai que pour la phase de calcul du projet HCMD qui a été faite entre le 19 Décembre 2006 et le 5 Juillet 2007, il ne faut pas pour autant en faire une vérité absolue pour toutes les grilles de type volontaire. Toutefois, nous pensons que ce facteur

est intéressant et établit pour la première fois à notre connaissance un point de comparaison.

4.4.2 Prévision pour la phase II du projet HCMD

Comme nous l'avons précisé dans la description du projet HCMD (chapitre 2), le projet ambitionne une deuxième phase de calcul sur un ensemble de 4000 protéines.

Essayons de prévoir le besoin de cette deuxième phase. Nous savons, d'après la description faite dans la section 4.3.1, que la quantité de calcul augmente en fonction du carré du nombre de protéines. Supposons que l'ensemble des 4000 protéines possède la même distribution des temps de calcul que les 168 protéines de la phase I. De plus, supposons que les nombres de points de séparation $N_{sep}(recepteur)$ ont aussi la même distribution. Alors la quantité de calcul à effectuer pour la phase II devrait être 566 fois plus importante.

Tout l'enjeu de la phase I était d'obtenir des informations sur une base de 168 protéines connues, afin de développer un algorithme prédictif (JET) permettant d'éliminer un ensemble de zones pour le calcul des énergies d'interaction. Les scientifiques du projet HCMD espèrent que la prédiction permettra de réduire le nombre de calcul d'énergie d'interaction d'un facteur 100.

Avec les hypothèses précédentes : distributions des temps de calcul et des points de séparation comparables et réduction d'un facteur 100, la quantité de calcul de la phase II devrait être 5,66 fois plus importante que celle de la phase I ($\frac{4{,}000^2}{168^2 * 100}$). Il est alors intéressant de se poser la question :

> *Combien de temps faudra-t-il à la plate-forme du World Community Grid pour calculer la phase II de projet HCMD ?*

En faisant l'hypothèse optimiste que la quantité de *processeurs virtuels équivalent plein temps* soit comparable aux périodes (B+C+D) de la phase I, nous pouvons espérer obtenir l'ensemble des résultats pour les 4000 protéines en 90 semaines, soit 1 an et 9 mois. Il est encore plus intéressant de se poser la question :

> *Combien de participants faudrait-il au World Community Grid pour être capable de faire l'ensemble des calculs de la phase II sur une période donnée ?*

Dans un souci de simplification, nous raisonnerons sur une période fixée à 10 mois (soit 40 semaines) et avec les statistiques actuelles de la grille World Community Grid.

Afin de répondre à cette question, nous allons d'abord raisonner en terme de *processeur virtuel équivalent plein temps*. Le temps processeur total pour la phase II, (en prenant le facteur multiplicatif 5,66 et le temps total consommé par la phase I) devrait être de plus de 457,48 siècles. Ce qui est équivalent à 59 730 *processeurs virtuels équivalent plein* pendant 10 mois.

Or, actuellement, le World Community Grid compte en moyenne 75 240 P_{vfp} pour 400 000 membres. Nous en déduisons donc qu'il est nécessaire d'avoir 5,33 membres pour obtenir un P_{vfp} ($\frac{400000}{75000}$).

De plus, nous savons que nous devrons partager ces processeurs avec au moins 4 autres projets. Nous pouvons donc espérer avoir 20 % des P_{vfp}, soit 15 048 P_{vfp}.

Au final, si nous souhaitons que le projet HCMD phase II se déroule en 10 mois, il faudrait donc être en mesure d'apporter 240 000 ($\approx (60000 - 15000) \times 5.33$ membres supplémentaires, qui seraient dédiés au projet HCMD, ou 1 200 000 ($\approx \frac{(60000-15000)\times 5.33}{20\%}$) nouveaux membres si ceux-ci ne calculent pas exclusivement pour le projet HCMD.

Beaucoup d'hypothèses optimistes sont faites pour arriver à ces nombres. Au moment de cette estimation, nous ne connaissons pas encore l'ensemble des 4000 protéines, nous ne pouvons donc pas vérifier les hypothèses sur les distributions des temps de calcul et le nombre de points de séparation. Ce résultat dépend aussi du facteur 100 d'amélioration apporté par l'algorithme prédictif (JET). Enfin, tout le raisonnement repose aussi sur le facteur 5,43 d'équivalence entre un processeur dédié et un P_{vfp}. Bref, le lecteur aura compris qu'il existe beaucoup de suppositions pour arriver à ces estimations. Néanmoins, elles permettent de prendre conscience des moyens qu'il faudra mettre en œuvre pour la phase II.

4.5 Conclusion

Ce chapitre a été l'occasion de présenter un ensemble d'expériences menées sur l'outil de recherche Grid'5000 qui a été mis en place par la communauté scientifique française. Nous nous sommes servis de cette plate-forme pour valider les fonctionnalités et le passage à l'échelle de l'intergiciel DIET. Nous avons montré qu'une plate-forme DIET est capable d'adresser plus de 1100 processeurs répartis sur la France tout en gardant des temps de réponse de l'ordre de la seconde. De plus, ces tests nous ont permis de révéler et d'améliorer le fonctionnement de DIET. Ils ont aussi initié le développement d'outils facilitateurs (GRUDU, *XmlGoDIETGenerator*) d'expériences dimensionnantes.

En outre, nous avons décrit l'ensemble des étapes préparatoires au lan-

cement d'une campagne de calcul qui a demandé au total plus de 80 siècles temps de calcul sur les machines des volontaires du World Community Grid. À partir du travail effectué lors de cette préparation nous avons proposé une étude détaillée et introduit le concept de *processeur virtuel plein temps* dans le but de mieux appréhender les grilles de volontaires. Ce concept a abouti à un point de comparaison entre une grille dédiée et la grille de volontaires. Enfin, nous avons proposé une extrapolation permettant d'estimer les moyens qu'il sera potentiellement nécessaire de mettre en place pour la seconde phase de calcul du projet HCMD.

Chapitre 5

Ordonnancement de DAG dans une grille

Sommaire

Dans ce chapitre, nous nous proposons d'étudier le problème d'ordonnancement à la volée de plusieurs applications de type graphe de tâches sur une grille de calcul à processeurs hétérogènes. Tout d'abord, nous commencerons par introduire le contexte de l'étude avec la présentation des *workflows* scientifiques, et la modélisation en graphe des applications et des plates-formes de calcul. Nous exposerons les différentes heuristiques utilisées classiquement pour résoudre le problème d'ordonnancement d'un graphe de tâches sur un ensemble de processeurs hétérogènes. Par la suite, nous préciserons le contexte original de l'étude qui semble à notre connaissance relativement peu étudié.

Le travail présenté ici est motivé par les applications issues du programme Décrypthon. Nous décrirons trois d'entre elles de façon à obtenir un graphe de tâches : c'est à dire un ensemble de programmes, fonctions ou méthodes qui s'enchaînent pour fournir des résultats. Les sommets du graphe représentent les programmes de l'application, et ses arêtes représentent les dépendances entre ces programmes.

Nous proposerons plusieurs heuristiques permettant l'ordonnancement dynamique de ces multiples applications sur une grille. Ces heuristiques permettent la prise en compte d'une priorité inter-applications et peuvent rendre l'exécution concurrente de celles-ci plus équitable. Nous dévoilerons l'architecture mise en place dans l'intergiciel DIET. Celle-ci s'articule autour d'un agent central ($\mathrm{MA_{DAG}}$) qui peut être dupliqué, rendant l'architecture robuste et passant à l'échelle. Nous terminons par une série d'expériences validant l'architecture et illustrant le comportement des heuristiques proposées sur un scénario d'exécution. Nous montrerons les avantages et les faiblesses des heuristiques.

Enfin, nous ouvrirons la discussion sur les perspectives de développement offertes par l'architecture proposée, et les travaux futurs qui pourront être entrepris à partir de celle-ci.

5.1 Introduction

Depuis quelques années déjà, les technologies de grille offrent aux utilisateurs l'accès à une multitude de services et de puissances de calcul pour traiter des données ou en générer de nouvelles. Ces perspectives ont donné un nouvel essor à la modélisation des applications et des services qui s'appuie sur l'utilisation des *workflows* (« flux de travail ») afin d'organiser l'utilisation des applications ou des services disponibles sur une grille informatique.

De manière générale, un *workflow* se définit comme : l'organisation et la formalisation de plusieurs tâches pour décrire une activité. Les *workflows*

sont utilisés dans un nombre important de domaines. Il existe les *workflows* de gestion de processus métier[1], c'est à dire l'organisation et la formalisation de plusieurs tâches pour décrire une activité. Ils se nomment aussi organigrammes de processus ou logigrammes. Ainsi, dans une entreprise, les processus métier se décrivent par la dépendance entre les activités :

- achat de produits : commande, paiement, livraison, *etc* ;
- prise de commandes : commande, livraison, paiement, *etc* ;
- création d'un produit : conception, réalisation, commercialisation, *etc* ;
- gestion de documents : acquisition, classement, stockage, diffusion, *etc* ;

Cette schématisation de l'activité permet de rationaliser et d'identifier les différentes étapes. Le traitement de l'information, et donc l'informatique, relève d'un *workflow*.

Le groupe *workflow management coalition* [48] propose la définition suivante :

> *La gestion des* workflows *est l'automatisation des processus métiers ou «* workflows *», pendant laquelle les documents, l'information ou les tâches sont transmises d'un acteur à un autre suivant des règles et des procédures établies.*

Cette rationalisation n'est pas le propre de l'informatique, elle se retrouve dans beaucoup de domaines. La modélisation est intimement liée à la notion d'ordonnancement des activités. En effet, une fois les processus définis et schématisés par un graphe, un ordre de traitement est établi : il est naturellement déterminé par le parcours du graphe suivant les arêtes.

Une des premières méthodes de suivi et d'organisation dans la gestion de projets est la méthode PERT[2]. Introduite dans les années cinquante dans la marine américaine, elle a permis de gérer la fabrication des missiles à ogive nucléaire Polaris. Cette technique consiste à mettre sous la forme d'un graphe les dépendances entre les activités d'un projet.

5.1.1 Les *workflows* scientifiques

En informatique, et plus particulièrement dans le calcul scientifique, une formalisation en graphe est aussi employée pour la modélisation d'applications venant de domaines aussi variés que les neurosciences, l'imagerie médicale, la physique des molécules, l'astronomie, ou encore la biologie cellulaire et moléculaire.

Nous désignons ces applications par des *workflows* scientifiques. Elles sont comme leurs homologues des processus métiers d'une entreprise ou la recette

[1] en anglais Business Process Management.

[2] l'acronyme de : **P**rogram (ou **P**roject) **E**valuation and **R**eview **T**echnique.

de cuisine d'un grand chef, une succession d'étapes qui permet l'étude d'un problème scientifique. Ce formalisme apporte plusieurs avancées pour l'utilisateur, il permet d'exprimer un mécanisme complexe d'analyse par un graphique simple, flexible et dynamique. Il décrit les interactions et la logique scientifique entre les différentes étapes. Lorsque ces *workflows* mettent en action une succession d'applications informatiques, ils expriment aussi le parallélisme intrinsèque de l'exécution des tâches qui seront exécutées par une ressource informatique (processeur).

Aussi, dans ce contexte, des domaines comme la bioinformatique s'organisent autour d'un portail internet collaboratif d'échange et de publication de *workflows* scientifiques [42] : `myExperiment.org`.

Il est important de noter qu'à ce stade un *workflow* peut avoir des étapes itératives, c'est à dire « refaire cette étape tant qu'on a pas le résultat voulu », ou des branches conditionnelles, c'est à dire « faire ce traitement si telle condition est réalisée », *etc.* Ainsi un langage de *workflow* s'apparente à un langage de programmation.

Le génie logiciel a poussé ce formalisme en introduisant l'UML (langage de modélisation unifié) qui permet de décrire graphiquement les données et les traitements effectués. Ce langage fournit au concepteur de logiciel un moyen d'exprimer toutes les dépendances, toutes les données et tous les traitements effectués dans un logiciel. Pas moins de 13 types de diagrammes différents existent pour conceptualiser un logiciel. Cependant ce formalisme bien que reconnu et standardisé par l'OMG (*Object Management Group*) reste dans le domaine du génie logiciel et de l'ingénierie. Il n'a pas encore sa place dans les domaines des sciences utilisant les *workflows* scientifiques même s'ils ont aussi une vocation visuelle de déroulement d'étapes, de méthodologie et d'algorithme de traitement.

5.1.2 Modélisation des applications

Une multitude d'applications informatiques se modélisent sous la forme d'un graphe orienté. La résolution d'un système linéaire $Ax = b$ est un cas d'école. A est alors une matrice triangulaire inférieure inversible de taille $n \times n$, et b un vecteur à n composantes. Il existe encore nombre d'applications d'imagerie travaillant sur une image et utilisant plusieurs algorithmes (programmes) différents pour analyser, filtrer et décomposer l'image. Le même genre de décomposition est appliqué pour les *workflows scientifiques*. Nous verrons par la suite la modélisation, sous la forme de graphe de tâches, trois des applications des projets du programme Décrypthon. L'application est décrite de manière à obtenir un ensemble de programmes, fonctions ou méthodes qui s'enchaînent pour fournir des résultats.

À ce stade, il est important de faire une distinction entre les différents types de *workflows* que nous retrouvons dans la littérature. Tristan Glatard propose dans sa thèse [40] une classification basée sur le contenu du langage d'expression d'un *workflow* :

- Fonctionnelle : la description de l'application est de haut niveau. Tous les types d'opérateurs de composition sont disponibles : boucles, conditions, *etc.* Les données en entrée des programmes ne sont pas forcément connues et, par exemple, le nombre d'itérations d'une boucle peut dépendre du résultat d'un programme. Tous les langages de programmation tombent dans cette catégorie.
- Services : comme la classe précédente la description est de haut niveau, cependant une information supplémentaire est ajoutée permettant de définir la ressource qui exécutera le programme.
- Tâches : la description de l'application est concrète, toutes les itérations et les conditions sont connues, il n'y a pas de boucles ni de conditions, l'ensemble des opérations est connu.
- Exécutable : comme la classe précédente, toutes les tâches sont connues, la ressource qui exécutera la tâche est aussi désignée.

Une cinquième classe d'expression d'un *workflow* est nommée « modèle formel ». Il permet de faire des analyses théoriques sur la description et le déroulement de l'application. Nous représentons les 4 classes d'expression

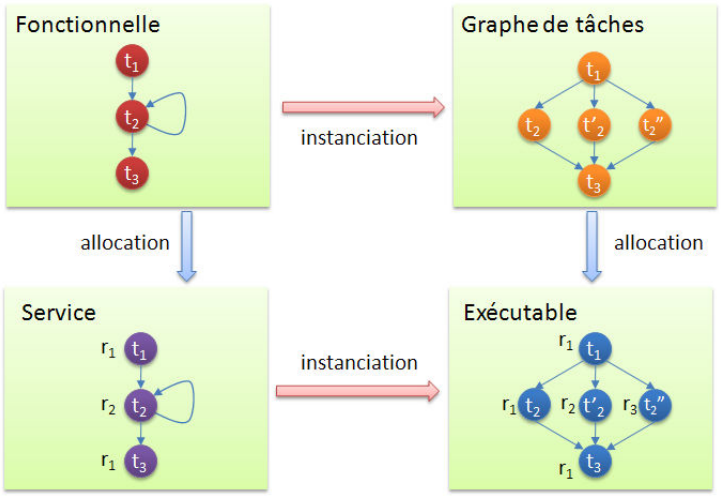


Figure 5.1 – Classification des *workflows*.

des *workflows* dans la figure 5.1. Il existe des opérations permettant de passer d'une classe à l'autre. Ainsi nous appellons *instanciation* l'opération qui

consiste à évaluer chaque boucle, opération de contrôle, valeurs de retour, *etc*, afin de connaître le nombre de tâches et les données transférées. L'opération de spécification des ressources est nommée *allocation*. Un exemple de passage entre les différentes classes de *workflows* est donné dans la figure 5.1

Ici, la modélisation sera restreinte à des graphes acycliques dirigés qui peuvent être appelés *workflows concrets*, ou aussi DAG (venant de l'anglais *Directed Acyclic Graph*), ou simplement *workflow* par abus de langage.

Soit un graphe valué $G(T, D, w, c)$ où :

- l'ensemble T des sommets représente les tâches ;
- l'ensemble D des arêtes représente les dépendances entre les tâches.
- la fonction de coût/poids d'un sommet $w : T \rightarrow \mathbb{R}$, cette fonction représente la quantité de travail nécessaire pour la tâche considérée.
- la fonction de coût/poids associée à une arête $c : E \rightarrow \mathbb{R}$, cette fonction représente le volume des données transférées entre deux tâches.

L'existence d'un lien $d_{i,j}$ entre deux tâches t_i et t_j traduit l'ordre partiel qui existe entre elles, c'est à dire que l'exécution de la tâche t_j ne peut commencer avant que l'exécution de t_j ne soit terminée et que les données soient transmises. Une tâche sans aucune arête entrante est appelée tâche d'*entrée* (t_1), une tâche sans aucune arête sortante est appelée tâche de *sortie* (t_3).

Cette modélisation met en évidence le parallélisme d'une application, ainsi dans l'exemple de la figure 5.1 on voit qu'un maximum de 3 tâches (t_2, t_2, t_2) peut être exécuté en parallèle après l'exécution de t_1.

5.1.3 Modélisation des ressources

Il existe plusieurs manières de définir les ressources d'une grille informatique. De façon générale, celles-ci sont aussi modélisées à l'aide d'un graphe dirigé valué où les sommets représentent les ressources de calcul et les arêtes le lien entre deux ressources.

Soit un ensemble de ressources R $\{r_u,\ 1 \leq u \leq |R|\}$ et un ensemble C de connexions entre deux ressources r_u and r_v, $c_{u,v}$ la capacité de ce lien, la forme du graphe $\wp = (R, C)$ représente la topologie de la plate-forme.

Il existe une quantité pléthorique de topologies : les chaînes linéaires, les bus de communication, les anneaux, les étoiles, les cliques, les arbres, les hypercubes, les mailles ou filets, et des combinaisons mixtes des différentes topologies énoncées.

La figure 5.2 donne un certain nombre d'exemples de ces topologies. Elles correspondent à une réalité matérielle d'implantation, par exemple le modèle bus de communication modélise le canal partagé sur une carte mère, la topologie étoile représente des ordinateurs raccordés par un commutateur.

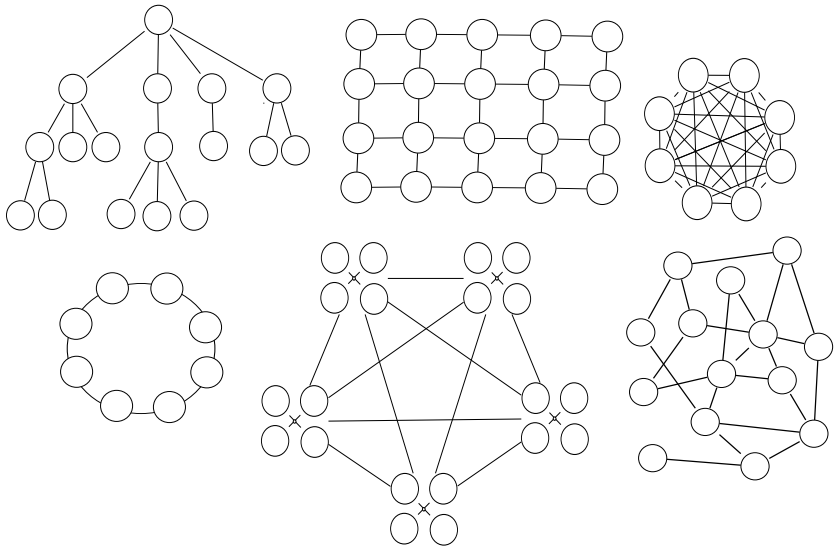


Figure 5.2 – Exemple de topologies.

Une fois la topologie de la plate-forme définie, il faut définir ses liens. Ils sont caractérisés par une fonction de coût associée au transfert d'une donnée. Cette fonction peut être plus ou moins élaborée suivant le degré de réalisme de la modélisation. Elle dépend du protocole qui est utilisé dans le réseau interconnectant les machines. Cette fonction de coût est le plus souvent basée sur la notion de bande passante qui dénote la vitesse de transfert d'une donnée et la notion de latence qui modélise le temps d'initialisation d'une communication. Nous définissons la manière de communiquer selon trois types de modèles de transfert :

- un port : une machine ne peut communiquer qu'avec une seule autre machine ;
- multi-port : une machine peut communiquer avec plusieurs machines à la fois ;
- multi-port borné : une machine peut communiquer avec plusieurs machines à la fois, la somme de toutes les communications ne peut pas excéder la bande passante totale de sortie ou d'entrée.

Les modèles de transferts peuvent être asymétriques ou symétriques, c'est à dire qu'une machine peut ne pas avoir les mêmes caractéristiques selon que l'on considère les communications entrantes ou sortantes (exemple : protocole ADSL). Les liens d'une plate-forme peuvent être aussi tous homogènes, ou différents (hétérogènes) suivant chaque lien considéré.

Ensuite, il convient de caractériser les sommets du graphe de la plate-

forme, c'est à dire les ressources de calcul, par une fonction de coût. Celle-ci se base sur la vitesse de calcul des processeurs, il serait également possible de la baser sur la quantité de mémoire. Comme pour les liens de communication, on définit aussi un modèle d'exécution d'une tâche :

- monotâche (une seule tâche à la fois)
- partagé : plusieurs tâches en même temps ;
- préemptif : une tâche peut être commencée puis arrêtée pour être remplacée par une autre puis reprendre ;
- non-préemptif : une fois la tâche commencée on ne peut pas l'arrêter sans avoir à la recommencer depuis le début.

Les ressources qui composent la plate-forme sont qualifiées :

- homogènes : toutes les ressources sont identiques, elles ont toutes les mêmes caractéristiques.
- hétérogènes uniformes : les ressources ont des caractéristiques différentes, mais il existe une relation entre les propriétés des ressources. Typiquement, ce processeur est deux fois moins rapide qu'un autre, celui-ci est 1,5 fois plus rapide, *etc.*
- hétérogènes non liées : les ressources ont des caractéristiques totalement différentes qui dépendent de la tâche qui leur est attribuée.

Étant donné qu'il existe plusieurs caractéristiques pour une machine de calcul (vitesse de calcul, mémoire, *etc*), il est possible d'imaginer, par exemple, des combinaisons dans l'homogénéité et l'hétérogénéité des vitesses et de la mémoire.

Ce qu'il faut retenir Certaines recherches ont pour but de définir une topologie et une fonction de coût réaliste pour les communications entre les machines [46, 55, 66], d'autres définissent un modèle réaliste d'exécution [34, 113] pour la durée des tâches sur une machine. Tous ces modèles sont compliqués, la plupart du temps dans les intergiciels de grille une abstraction simplifiée des ressources est employée :

Une topologie en clique virtuelle de ressources, c'est à dire que toutes les machines peuvent communiquer entre elles (graphe complet de machines). Cette topologie ne rend pas compte de l'implantation réseau sous-jacente, cependant elle relate bien l'état de fait retrouvé dans certaines grilles où toutes les machines peuvent communiquer entre elles.

On utilise ensuite des fonctions de coût d'exécution et de communication : homogène, hétérogène uniforme et hétérogène non liée. Ici, nous considérerons le cas général hétérogène non liée. L'idée, dans les fonctions de coût, sera de toujours rester au plus proche de la réalité, cependant, il est souvent difficile de reproduire et de prévoir totalement une

plate-forme grille tellement les couches logicielles et protocolaires sont nombreuses et se superposent.

5.1.4 Ordonnancement et l'allocation de ressources

L'ordonnancement d'un graphe de tâches sur une plate-forme de grille pose la question de l'attribution d'une date de début et d'une allocation de ressources pour l'exécution des tâches qui composent le graphe. Cet ordonnancement doit respecter les contraintes de précédence définies par les liens dans le DAG. La difficulté dans le choix de la machine et dans l'ordonnancement des tâches réside dans la prise en compte à la fois des tâches à exécuter et des transferts de données entre les tâches.

Le problème de décision associé au problème d'ordonnancement d'un graphe de tâches sur un ensemble fini de machines est NP-complet [37]. En termes simples la théorie de la complexité pour les problèmes informatiques définit la difficulté de trouver une solution qui satisfasse les contraintes pour l'objectif fixé. En particulier, la classe NP est comme la classe des problèmes dont on peut vérifier les solutions en temps raisonnable (polynomial en la taille des données). Parmi ceux-là, on distingue les problèmes NP-complet, dont il est impossible de trouver la ou les solutions exactes autrement qu'en les énumérant et en les vérifiant toutes[3]. Une manière, plus informelle pour définir la classe NP : c'est la classe des problèmes pour lesquels les seuls algorithmes connus de résolution sont ceux ayant une complexité exponentielle en la taille des données.

Il existe un formalisme de classification des problèmes dans le domaine de l'ordonnancement des problèmes d'atelier (en anglais *job-shop, flow-shop, Open-shop* ...). L'ordonnancement d'atelier correspond au domaine qui s'intéresse aux problèmes d'optimisation des usines de confection de pièces ou de construction. Cette classification, issue des ouvrages de référence sur la théorie de l'ordonnancement [26, 54], et de l'article [45], introduit une notation à trois paramètres ($\alpha \mid \beta \mid \gamma$). Ils décrivent les instances du problème d'ordonnancement.

- α décrit les ressources du problème ;
- β décrit les contraintes et les caractéristiques du système ;
- γ décrit l'objectif fixé dans les critères d'optimisation.

Nous renvoyons le lecteur intéressé au livre de Peter Brucker [18] et au site internet de l'auteur qui référence les problèmes étudiés et les résultats sur la complexité de leur résolution.

En utilisant la notation ($\alpha \mid \beta \mid \gamma$), le problème d'ordonnancement d'un

[3] sous réserve que $P \neq NP$, mais laissons ça aux théoriciens.

graphe de tâches sur une plate-forme grille hétérogène se note par (Qm $|prec, c\ |C_{max}$) ou (Rm $|prec, c\ |\ C_{max}$) suivant les applications et les ordinateurs. Dans ces deux notations $prec$ exprime la contrainte de précédence, c exprime les contraintes de communication entre les tâches, C_{max} correspond au critère d'optimisation recherché, ici, la minimisation du temps de terminaison de l'application (*makespan*). Enfin Qm et Rm représentent respectivement un ensemble de m machines uniformément hétérogènes et un ensemble de m machines dont les temps d'exécution dépendent des tâches exécutées et de la machine sélectionnée.

Cette notation est peu employée dans la littérature traitant de l'ordonnancement dans les grilles de calcul. Elle est employée de manière sporadique dans les travaux sur l'ordonnancement d'une plate-forme maître-esclave [62, 73]. En effet, la notation ne permet pas d'exprimer de manière concise les contraintes sur la topologie de connexion entre les ressources et le modèle du transfert entre les ordinateurs. De plus, ce formalisme convient d'avantage à une étude théorique pure, il est moins bien adapté aux modèles spécifiques utilisés dans la communauté de l'ordonnancement des grilles informatiques.

5.2 Les différentes heuristiques d'ordonnancement

Pour résoudre un problème d'ordonnancement NP-complet, des heuristiques (ou intuitions) sont utilisées, elles permettent de s'approcher d'une solution acceptable en faisant certains choix. Il existe un nombre pléthorique d'heuristiques développées dans le domaine de l'ordonnancement de graphe de tâches incluant les méthodes de *branch and bound* (séparation et évaluation), programmation dynamique, algorithme génétique. Nous présenterons ici le principe de 4 classes [59] d'heuristiques d'ordonnancement de graphe de tâches et quelques références vers celles-ci :

- heuristique de liste ;
- heuristique de groupement de tâches ;
- heuristique de duplication de tâches ;
- méta-heuristique.

Il existe de nombreux articles [19, 20, 25, 114] destinés au recensement des heuristiques et à leur comparaison. Elles sont souvent délicates à comparer car elles n'ont pas forcément été développées dans un cadre identique et peuvent être spécialisées pour un type d'application (topologie du DAG), pour un type de plate-forme (topologie de plate-forme), *etc.*

5.2.1 Les heuristiques de liste

Les heuristiques de liste maintiennent une liste de tâches triées suivant une priorité. Ces heuristiques sont toutes basées sur la succession des deux étapes suivantes :

1. Prendre une tâche parmi celles qui sont prêtes et/ou pas encore allouées à une machine. Une tâche devient prête lorsque toutes les tâches parentes sont terminées et les données nécessaires aux calculs sont disponibles.
2. Sélectionner une machine pour exécuter la tâche et allouer celle-ci à la machine.

La première étape est aussi nommée phase de *prioritisation* car elle permet de faire un choix lorsqu'il existe plusieurs tâches disponibles. La priorité attribuée aux tâches permet de donner un ordre d'exécution. Cet ordre est déjà induit par l'ordre partiel défini par la relation de précédence qui se retrouve avec une traversée topologique du graphe de tâches. Deux types d'algorithmes de listes se distinguent. Les heuristiques à priorité statique : la priorité pour la tâche est établie une fois pour toutes, et ne sera jamais remise en cause au cours de l'heuristique. À l'inverse, les heuristiques dynamiques réévaluent la priorité d'une tâche au fur et à mesure de l'exécution du DAG. Il y a différentes façons d'établir une priorité sur les tâches. Les valeurs fréquemment utilisées pour construire une métrique de priorité sont :

- *t-level* : correspond à la longueur d'un plus grand chemin depuis le nœud d'entrée jusqu'à la tâche considérée t_i. La longueur du chemin étant définie par la somme des valuations des tâches et des arêtes en excluant la valeur de la tâche t_i. Parfois, cette valeur est appelée ASAP[4] car elle correspond à la date de commencement au plus tôt de la tâche.
- *b-level* correspond à la longueur d'un plus grand chemin depuis la tâche t_i jusqu'au nœud de sortie.
- *s-level* correspond comme pour le *b-level* à la longueur d'un plus grand chemin jusqu'au nœud de sortie sans compter la valuation des arêtes.
- ALAP[5] correspond à la différence entre la longueur d'un chemin critique et la valeur du b-level du nœud considéré. C'est aussi la date « au plus tard » de la tâche.

À partir de l'exemple de la figure 5.3, l'ensemble des valeurs *t-level*, *b-level*, *s-level* et ALAP est donné dans le tableau 5.1.

La deuxième étape est la phase d'*allocation*. Il faut choisir le processeur qui exécutera la tâche sélectionnée à l'étape précédente. Là encore les critères

[4]As Soon As Possible
[5]As Late As Possible

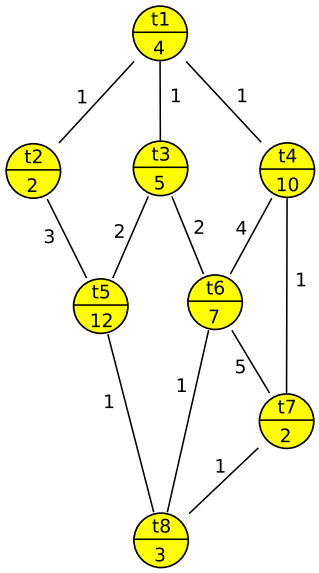


Figure 5.3 – Exemple d'un graphe de tâches.

Nœeud	*b-level*	*t-level*	*s-level*	ALAP
t_1	37	0	26	0
t_2	21	5	17	16
t_3	25	5	20	12
t_4	32	5	22	5
t_5	16	12	15	21
t_6	18	19	12	19
t_7	6	31	5	31
t_8	3	34	3	34

Tableau 5.1 – Attributs correspondant à l'exemple du DAG de la figure 5.3.

de choix sont nombreux. Les principaux répertoriés [16] sont min-min, sufferage, max-min, MCT[6], SRPT[7], LRPT[8], *etc.* À ce stade, il faut allouer une tâche sur une plate-forme hétérogène, compte tenu des choix faits auparavant.

Les heuristiques de liste sont, la plupart du temps, utilisées car une garantie sur la qualité de l'ordonnancement a été démontrée. En effet, dans le cas des processeurs homogènes, et sans considérer les communications entre les

[6]Minimum Completion Time
[7]Shortest Remaining Processing Time
[8]Longest Remaining Processing Time

tâches du DAG, les travaux de Graham [44] montrent que toute heuristique de liste obtient un ordonnancement dont la durée totale d'exécution est, au pire, à 50 % de l'ordonnancement optimal (que l'on ne connaît pas). On parle de rapport de compétitivité, il est exactement égal à $2 - \frac{1}{p}$ avec p le nombre de processeurs homogènes disponibles.

Dans le cas hétérogène, en considérant les temps de communication, cette garantie ne tient plus, mais l'on obtient néanmoins de bonnes performances et plusieurs articles l'ont vérifié par simulation [12, 19].

Étant donné que les processeurs de la plate-forme visée sont hétérogènes, cela n'a pas vraiment de sens de considérer la valuation donnée dans les tâches du graphe. Généralement cette valeur est remplacée par la moyenne des temps d'exécution sur tous les processeurs de la plate-forme. Une multitude d'heuristiques de liste ont été proposées, pour en citer les principales : Heterogeneous Earliest Finish Time (HEFT) [110, 111], Critical Path on Processor (CPOP) [110, 111], Levelized Min-Time (LMT) [50], Generalized Dynamic Level (GDL) [87], Iso-Level Heterogeneous Allocation (ILHA) [12], SDC [86]. Par la suite, nous détaillerons en particulier l'heuristique HEFT qui est l'heuristique à partir de laquelle nous avons conçu les heuristiques que nous proposerons.

5.2.2 Groupement de tâches

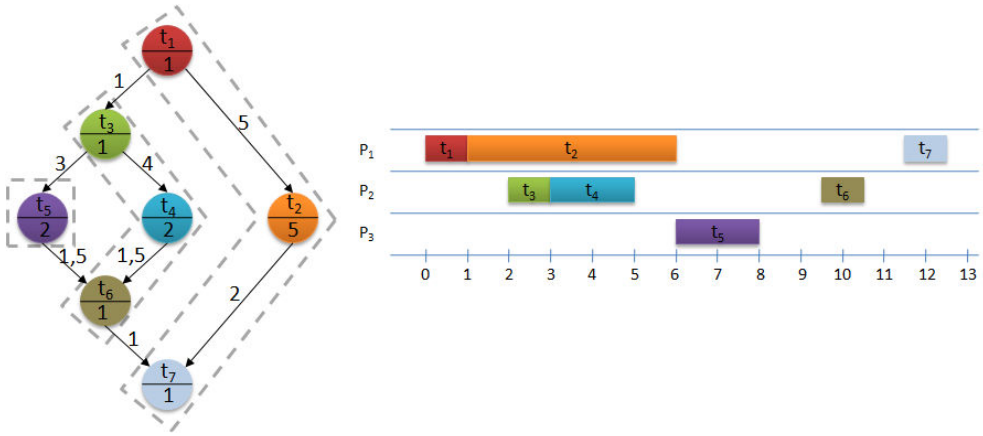


Figure 5.4 – Exemple$_1$ de groupement de tâches.

Les heuristiques de groupement de tâches (*clustering algorithms*) reposent sur l'idée de regrouper des tâches ensemble dans le but de supprimer les communications trop coûteuses. Les algorithmes fonctionnent en deux phases :

– la première phase groupe les tâches du DAG. Le groupement est effectué par une succession d'étapes de raffinement sans retour en arrière : une fois les tâches groupées, le groupement d'une étape précédente n'est pas remis en cause. Une méthode classique consistera à rassembler deux tâches qui s'échangent une grande quantité de données (i.e. la valuation de l'arête est grande). Cette phase permet d'obtenir un nouveau DAG dont les nœuds représentent cette fois un groupement de tâches.
– la deuxième phase consiste à trouver un ordonnancement et une allocation sur un processeur pour les groupes de tâches construits à l'étape précédente. Ainsi, aucune communication inter-machine ne sera faite au sein d'un groupe puisque qu'elles seront toutes exécutées par la même machine.

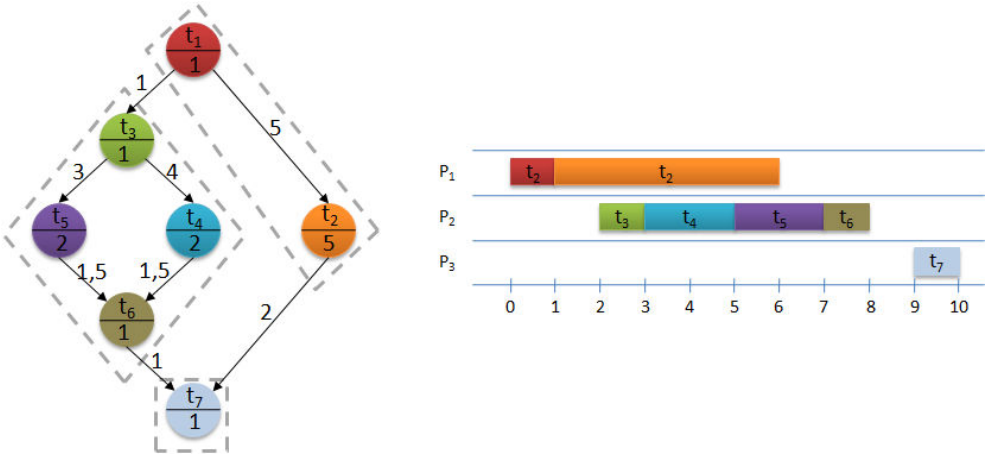


Figure 5.5 – Exemple$_2$ de groupement de tâches.

Le principe de ces heuristiques repose donc sur la diminution du nombre de sommets dans le DAG en formant des groupes. On parle de réduction de la granularité du graphe de tâches. Ce genre de technique est utilisé lorsque le nombre de tâches dans le DAG est important, et lorsque l'on désire réduire le volume de données échangées entre les processeurs. Cela permet de plus de diminuer la complexité lors de l'ordonnancement des tâches sur les machines. Les figures 5.4 et 5.5 montrent deux exemples où les tâches ont été groupées avec deux stratégies différentes. Là encore la littérature est abondante, elle est toutefois souvent restreinte au cas homogène [39, 56]. Parmi les heuristiques les plus souvent citées : Linear Clustering Method [56], Edge Zeroing algorithm [82] et Dominant Sequence Clustering(DSC) [115].

5.2.3 Duplication de tâches

Les heuristiques à duplication de tâches consistent à allouer de manière redondante certaines des tâches « importantes » dont d'autres dépendent. Le but recherché est donc de réduire le temps avant que les tâches en attente puissent commencer ce qui peut éventuellement améliorer le temps d'exécution global de l'application.

Afin de bien comprendre l'utilité de ces heuristiques, il est présenté dans la figure 5.6 un exemple d'ordonnancement d'une application sur 2 machines (par simplicité nous supposons que les machines et les liens de communication sont homogènes). La valuation des tâches représente leur durée d'exécution, et celle des arêtes représente la durée de transfert des données entre les deux tâches. Ici, le temps de terminaison de l'application est de 13 unités de temps.

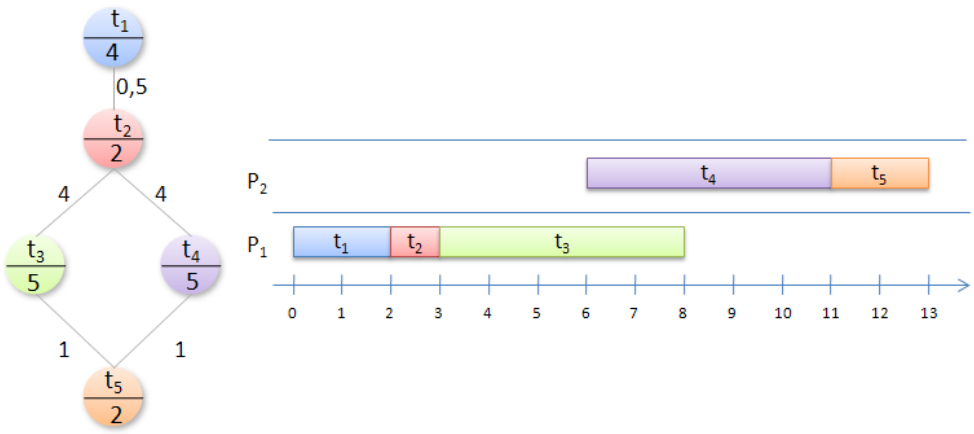


Figure 5.6 – Exemple d'ordonnancement d'un graphe de tâches sur 2 processeurs.

Dans l'ordonnancement précédent, beaucoup de temps est perdu à envoyer les données en sortie de la tâche t_2 à la machine qui exécute la tâche t_4. Si nous dupliquons la tâche t_2 (de sorte que la machine p_2 n'ait pas à attendre le transfert de la donnée issue de t_2, puisqu'elle aura été effectuée aussi sur cette machine), tout se passe comme si nous avions désormais le graphe de tâches présenté dans la figure 5.7. La durée d'exécution de l'application sera alors réduite à 10,5 unités de temps.

Dans l'absolu, si toutes les tâches hors chemin critique sont dupliquées, on obtient un ensemble de graphes de tâches de type chaîne sans dépendance entre elles. Ces techniques [1, 2, 11] ont été conçues pour des environnements

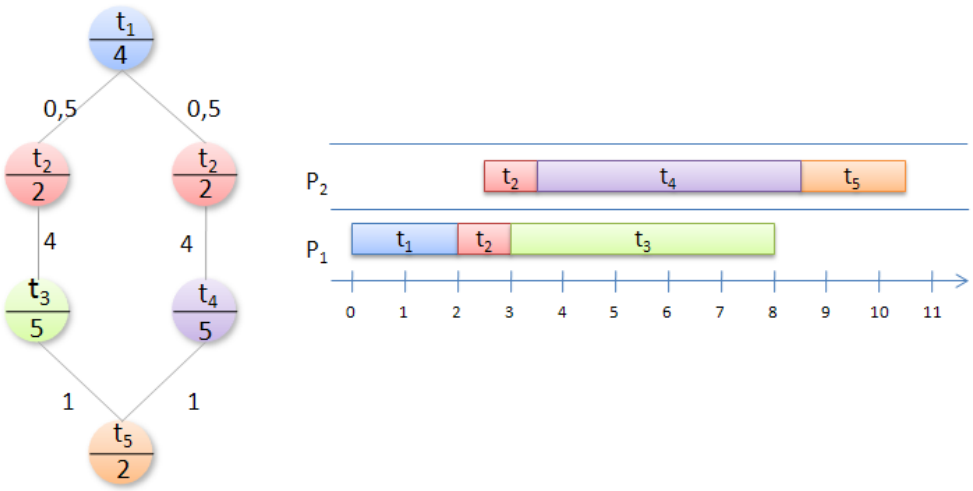


Figure 5.7 – Exemple avec duplication sur 2 processeurs.

où le nombre de ressources est important.

5.2.4 Méta-heuristiques

Le terme de méta-heuristiques englobe toutes les techniques qui recherchent dans l'espace des solutions possibles pour l'ordonnancement du graphe de tâches sur une plate-forme grille.

Ces méthodes sont souvent utilisées face à des problèmes NP-Complet où la meilleure solution ne peut être trouvée autrement qu'en explorant toutes les solutions possibles. L'idée est donc d'explorer de manière « intelligente » l'espace des solutions afin de trouver une meilleure solution en fonction de l'objectif. La différence entre les méthodes réside dans « l'intelligence » d'exploration des solutions possibles. Bien que ce genre de techniques soient très intéressantes et donnent souvent de bons résultats, elles ne sont pas adaptées à des problèmes où l'on doit trouver un ordonnancement rapidement. En effet, plus la taille du problème (nombre de tâches, nombre de dépendances, nombre de ressources) augmente, plus l'espace des solutions à explorer est grand. De plus, il faut relancer une recherche à chaque modification du problème initial.

À titre d'illustration on citera les travaux de Memaz et al. [68] qui s'est servi de la grille de calcul Grid'5000 pour résoudre le problème d'ordonnancement d'atelier flow-shop Ta56 (50 travaux sur 20 machines). Il aura fallu plus de 22 ans de temps processeur cumulé pour trouver la solution exacte au

problème de minimisation Ta56. Les principales méthodes [79] utilisées dans ce domaine sont la programmation par contrainte logique, les algorithmes génétiques, la recherche tabou et le recuit simulé.

5.3 Tour d'horizon des gestionnaires d'exécution de *workflow*

Il existe beaucoup de gestionnaires d'exécution de *workflows* pour les grilles de calcul. À vrai dire, la plupart des grilles actuellement utilisées propose un système permettant de prendre en compte des dépendances entre des travaux à exécuter. Nous décrivons succintement quelques uns d'entre eux :

Pegasus/DAGMan [28, 92] est un gestionnaire d'exécution de *workflows* qui s'appuie sur le module DAGman [92] de l'intergiciel Condor-G [36] pour l'allocation des tâches sur les ressources informatiques. Celui-ci travaille principalement sur la structure du graphe de tâches afin de le réduire pour permettre une exécution efficace. Il utilise donc d'abord un partitionnement du graphe de tâches en sous-groupes de tâches (heuristiques de *clustering* présentées auparavant). Les sous-groupes de tâches sont alors soumis à DAGman pour l'exécution. DAGman place les tâches dans une liste et alloue les calculs prêts aux machines disponibles. Lorsqu'une tâche se termine, ses tâches filles sont à leur tour démarrées.

MOTEUR [40, 41] est un gestionnaire de flots de calcul. Il permet la description de flots applicatifs complexes avec un schéma de composition de données dans un formalisme très compact. La prise en compte transparente du parallélisme de services et de données est exprimé implicitement en séparant le graphe de flot et les données. MOTEUR implante des interfaces pour les services standards de type service Web ou GridRPC.

GridAnt [6] est un système de gestion des *workflows* côté client. Il a été conçu en 2002 à l'Argonne National Laboratory pour les utilisateurs de grilles de calcul dans le but d'être un outil pratique pour exprimer (spécifier des pré-conditions et des tâches exécutables en parallèles) et contrôler les séquences d'exécution. GridAnt ne fournit pas de mécanisme d'allocation automatique des ressources, l'utilisateur doit lui-même spécifier les machines qui exécuteront les travaux contenus dans son *workflow*.

Triana [67] est un environnement permettant la construction et l'exécution de *workflows*. Il est développé à l'Université de Cardiff pour le calcul distribué grâce aux services web. Il inclut le support de GridLab [84]

GAT (Grid Application Toolkit) pour la soumission de *jobs*, la gestion des données, des fichiers et de la sécurité. Triana est surtout un outil permettant la création d'applications à partir des services disponibles sur la grille qu'il adresse. Le moteur d'exécution qu'il utilise permet d'appeler un service sur une machine spécifiée par l'utilisateur. Il déroule donc le scénario préetabli par l'utilisateur en respectant les contraintes de dépendance. Il ne fait pas à proprement parler d'allocation ni d'ordonnancement dans le choix des travaux à effectuer.

GridFlow [22] est un gestionnaire d'exécution de *workflows* développé à l'Université de Warwick, il s'appuie sur le gestionnaire de grille ARMS [21] (Agent-based Resource Management System for grid Computing) et sur TITAN le gestionnaire d'exécution de tâches sur des ressources locales. Il intégre une interface permettant de construire dynamiquement le graphe de dépendances entre plusieurs activités. Une activité peut elle-même représenter une application avec des dépendances (*sub-workflows*). L'allocation des activités qui composent le *workflow* s'effectue en simulant l'exécution à partir des données recueillies par le moniteur de ressources (MDS). L'exécution est gérée à deux niveaux, le module ARMS s'occupe du *workflow* global tandis que le module TITAN gère l'exécution des activités. La technique globale d'ordonnancement et d'allocation s'apparente à une heuristique de liste. L'allocation des activités qui composent le *workflow* peut aussi s'effectuer sous la contrainte de l'utilisateur.

Taverna [100] est le gestionnaire de *workflows* du projet ^{my}Grid. Il propose une interface graphique pour composer son application à partir de services Web. Il permet de définir un graphe d'appel à des services qu'il composera avec une description des données sur lesquelles le même graphe de services sera appelé. Il existe des opérateurs de compositions de données (*all-to-all, one-to-one*) ajoutant un formalisme permettant d'exprimer le parallélisme sur les données d'entrée et de sortie des services. Dans ce gestionnaire de *workflows* de services, il n'y a pas de stratégie d'ordonnancement et d'allocation globale du graphe de tâches sur les ressources étant donné qu'elles sont déjà définies avant l'appel. Cependant les invocations peuvent être exécutées de manière concurrente si les services utilisés supportent des appels concurrents.

ASKALON [33] est un projet développé par l'Université d'Innsbruck. Il permet à un développeur désirant utiliser les ressources d'une grille de composer son application à partir d'une interface graphique. ASKALON propose deux modes de représentation des *workflows*. Le premier, basé sur l'expression fonctionnelle avec le langage (AGWL), et

l'autre, basé sur les graphes de tâches (CGWL). L'ordonnancement proposé par ce gestionnaire de *workflows* s'appuie sur des heuristiques de listes avec la possibilité de définir différentes politiques de sélection d'une tâche prête et d'une ressource adéquate. Ils ont aussi implanté des méta-heuristiques de type « recuit simulé » pour explorer l'espace des solutions possibles pour l'allocation des ressources aux tâches.

Une taxinomie est donnée par Buyya et al [116] des différents gestionnaires de *workflows*. La classification (voir la figure 5.1) que nous avons exprimée au niveau des langages d'expression des *workflows* se retrouve avec les gestionnaires d'éxécution : certains offrent la possibilité de définir des *workflows* exprimés sous la forme abstraite c'est à dire sans spécifier les données (expression fonctionnelle ou expression de services). Ils doivent dans ce cas instancier leur *workflow* avec les données pour être en mesure de l'exécuter. D'autres gestionnaires expriment leur *workflow* sous forme concrète (graphe de tâches i.e DAG). Ils appliquent alors une des heuristiques que nous avons évoquée (duplication de tâches, groupement de tâches, heuristique de liste). Enfin, un dernier groupe l'exprime sous forme concrète avec une pré-allocation des ressources. Dans ce cas, le gestionnaire de *workflows* peut être vu comme une application dont les décisions d'ordonnancement sont fixées par l'utilisateur qui connait précisément l'infrastructure des ressources qu'il utilise.

Ce qu'il faut retenir : Parmi tous ces gestionnaires d'exécution d'application avec dépendances, lorsque l'allocation n'est pas faite par l'utilisateur, tous utilisent une stratégie qui s'apparente à un algorithme de liste pour l'ordonnancement et l'allocation. Ils peuvent faire une allocation nommée « juste à temps », c'est à dire choisir une ressource au moment où toutes les dépendances de la tâche sont satisfaites [22, 33, 67]. Il est éventuellement appliqué des stratégies de groupage de tâches lorsque le nombre et les spécificités des ressources qu'il adresse le nécessite [28, 41]. Plusieurs études tendent à montrer par simulation, et appuyées par des expériences, qu'il est intéressant d'utiliser l'heuristique de liste HEFT [33, 110, 111].

5.4 Contexte multi-applications et multi-utilisateurs

Ici, nous nous intéressons au problème très concret, d'ordonnancement à la volée de plusieurs graphes de tâches sur une grille hétérogène de calcul. Dans un contexte où il existe :

– plusieurs utilisateurs ;

– des applications différentes (graphes de tâches) pour chaque utilisateur ;
– un partage des ressources de la grille pour les calculs.

5.4.1 Modélisation

Nous décrirons les applications comme précédemment en ajoutant le fait qu'il y a plusieurs graphes de tâches, soit :

* $(a_1, a_2, a_3, ...)$ un ensemble $\mathcal{A}$ d'applications.
* pour chaque application a_n est associée à un temps d'arrivée r^{a_n}.
* $\forall a_n \in \mathcal{A}, G^{a_n} = (T^{a_n}, D^{a_n})$ le graphe acyclique associé à chaque instance a_n.
* T^{a_n} : les tâches de l'application a_n, $(t_i^{a_n})_{i\in[,|T^{a_n}|]}$.
* D^{a_n} : les dépendances entre les tâches $t_i^{a_n}$ et $t_j^{a_n}$, $(d_{i,j}^{a_n})_{(i,j)\in[1,|T^{a_n}|]}$.
* $w_i^{a_n}$: le coût associé à la tâche $t_i^{a_n}$.
* $\delta_{i,j}^{a_n}$: le coût associé à la dépendance entre $t_i^{a_n}$ et $t_j^{a_n}$.

Pour mémoire, la modélisation des ressources est reprise ci-dessous :

* $G = (M, L)$ le graphe complet des ressources composant la plate-forme grille.
* M : l'ensemble des machines (processeurs), $(m_i)_{i\in[1,|M|]}$.
* L : l'ensemble des liens réseaux entre les machines m_i et m_j, $(l_{i,j})_{(i,j)\in[1,|M|]}$.
* ρ_i : la puissance de traitement de la machine m_i.
* $\gamma_{i,j}$ la capacité du lien de communication entre la machine m_i et m_j.

Les coûts de calcul

Si la plate-forme est hétérogène uniforme, soit une tâche $t_i^{a_n}$ avec le coût w_i, exécutée sur la machine p_j avec une vitesse ρ_j, le coût de son exécution est :

$$w_i^{a_n}.\rho_j$$

Si la plate-forme est hétérogène non liée, la vitesse de calcul pour chaque processeur dépend de la tâche considérée. Soit une tâche $t_i^{a_n}$ avec le coût w_i, exécutée sur la machine p_j, le coût de son excution est :

$$w_i^{a_n}.\rho_j(t_i^{a_n})$$

Les coûts de communication

Nous considérons les liens de communication hétérogènes liés. Soit la donnée $d_{i,j}^{a_n}$ entre les tâches $t_i^{a_n}$ et $t_j^{a_n}$ de l'application a_n transférée entre les machines p_k et p_m sur le lien $l_{k,m}$, le coût du transfert est défini par :

$$\delta_{i,j}^{a_n}.\gamma_{k,m}$$

Nous considérons aussi que le coût de transfert sur une même machine $l_{i,i}$ est nul. En effet la donnée étant déjà présente sur la machine, il n'y a pas de transfert à effectuer.

Ces notations nous permettrons de faire les calculs pour l'établissement des priorités des tâches inter et intra-applications. Elles servent aussi à déterminer les temps de terminaison, au plus tôt, nécessaires dans les heuristiques que nous utiliserons.

5.4.2 Heuristiques multi-applications

Il existe une variété impressionnante d'heuristiques permettant l'ordonnancement d"un seul graphe de tâches sur des machines hétérogènes. Peu d'articles traitent de l'ordonnancement de plusieurs graphes de tâches dans un contexte multi-utilisateurs à des dates d'arrivée différentes. En effet l'article de Zhao et al. [119] présente plusieurs heuristiques de liste, cependant les auteurs supposent que les graphes de tâches sont tous soumis au même moment ($\forall a_n \in \mathcal{A}, r^{a_n} = 0$), de plus ils se limitent à une étude par simulation de leurs heuristiques.

Dans une autre série d'articles Iverson et al. présentent l'heuristique LMT [50] et une architecture décentralisée [49] pour ordonnancer plusieurs applications venant de plusieurs utilisateurs. Cependant l'heuristique présentée ne permet pas un ordonnancement inter-DAG et l'architecture proposée est restée au stade de la simulation.

Duan et al. [32] présentent une formulation du problème d'ordonnancement de plusieurs *workflows* comportant un grand nombre d'activités identiques, comme un problème d'ordonnancement de tâches indépendantes. Ils introduisent deux heuristiques basées sur la théorie des jeux et les comparent aux heuristiques classiquement utilisées pour des tâches indépendantes [16](min-min, sufferage, max-min, olb, met). Ils concluent que leurs heuristiques sont une bonne approche si les applications sont exprimées sous la forme d'un jeu.

Dans la suite de cette section, nous présentons une série de 6 heuristiques que nous avons conçues, permettant de prendre en compte l'arrivée de plusieurs graphes de tâches à des dates quelconques. Nous verrons comment nous avons mis en œuvre ces heuristiques dans l'intergiciel DIET et comment nous avons tiré partie des fonctionnalités de ce dernier. Enfin, nous présenterons les résultats d'expériences réalisées à partir de divers scénarii avec 3 types d'applications tirées des applications du programme Décrypthon.

5.4.3 Principe de base

In fine, les heuristiques de liste sont pratiquement les seules techniques employées dans les intergiciels de grille. À ceci deux raisons principales : elles donnent des résultats satisfaisants et elles sont assez « faciles » à implanter. D'autre part, même si l'on utilise une heuristique de groupement [82, 56, 115] ou de duplication [1, 2, 11], cette modification portera sur la structure du DAG avant l'ordonnancement, mais le processus de sélection des ressources exécutant les tâches reste basé sur un ordonnancement de listes. Enfin, les algorithmes dits méta-heuristiques ne sont pas applicables à des plates-formes dynamiques du fait de leur temps de réponse avant de produire un ordonnancement [68].

Le principe de base que nous allons exploiter dans toutes nos heuristiques est basé sur un algorithme très simple.

pour chaque *nouveau DAG d* **faire**
  (a)Calculer une priorité pour chaque tâche;
  Trier les tâches du DAG par ordre de priorité décroissante;
  tant que *il reste des tâches non exécutées* **faire**
    (b) Sélectionner une tâche prête t_r;
    (c) Chercher une machine p pour la tâche t_r;
    Allouer la tâche t_r sur la machine p;
  fin
fin

Algorithme 2 : Algorithme général d'ordonnancement des graphes de tâches.

Dans l'algorithme 2, trois étapes ne sont pas définies :
(a) Calculer la priorité de chaque tâche ;
(b) Selectionner une tâche t_r prête ;
(c) Chercher une machine p pour la tâche t_r.
Ces trois étapes sont déterminantes dans l'obtention d'un ordonnancement. Elles font la différence entre les différentes heuristiques de liste [86, 118].

Ici, nous nous intéressons plus particulièrement à la sélection de la tâche prête (étape (b)). En effet, il peut exister dans le système plusieurs applications (DAG) en train d'être exécutées à un instant donné. Il est nécessaire de définir un ordre entre les différentes tâches prêtes de ces applications. Nous avons développé 5 méthodes de sélection Le principe est de maintenir une double priorité sur les tâches :

- la première priorité sera celle inter-DAG ;
- la deuxième priorité sera celle intra-DAG.

Ainsi, la sélection de la tâche se fera d'abord par rapport à la priorité inter-DAG. S'il y a égalité, la priorité intra-DAG est utilisée. Dans l'ensemble des heuristiques, la priorité intra-DAG reste inchangée, nous utilisons celle définie dans l'heuristique HEFT, à savoir :

$$\forall a_n \in \mathcal{A}, rank_{intraDAG}(t_i^{a_n}) = rank_{HEFT}(t_i^{a_n})$$

$$rank_{HEFT}(t_i^{a_n}) = \overline{w_i}^{a_n} + \max_{t_k^{a_n} \in Succ(t_i^{a_n})} (\overline{\delta_{i,k}}^{a_n} + rank_{HEFT}(t_k^{a_n})$$

Avec $t_i^{a_n}$ une tâche de l'application a_n considérée, $\overline{w_i^{a_n}}$ le temps moyen d'exécution de $t_i^{a_n}$ sur toutes les machines disponibles , $\overline{\delta_{i,k}}^{a_n}$ le coup moyen de transfert des données entre $t_i^{a_n}$ et $t_k^{a_n}$ et $Succ(t_i^{a_n})$ l'ensemble des successeurs de t_i dans le graphe de tâches. Ce qui correspond au *t-level* que nous avons défini auparavant en utilisant les temps moyens de calcul et les temps moyens de communication sur la plate-forme considérée. Le choix d'utiliser la méthode de l'heuristique HEFT pour définir la priorité intra-DAG est motivé par les bonnes performances obtenues par cette méthode dans les cas généraux [118].

G-HEFT

Nous avons baptisé cette méthode G-HEFT (Global-HEFT) car son principe est de sélectionner une tâche prête parmi l'ensemble des tâches de chaque DAG en se basant uniquement sur la priorité définie initialement au sein d'un DAG. Tout revient à considérer l'ensemble des DAGs comme un seul DAG virtuel construit à partir des autres, en ajoutant une tâche fictive (de coût nul) en entrée et en sortie. Ce qui revient à définir la priorité des tâches dans le DAG virtuel (agrégation de tous les autres) en utilisant le rang HEFT pour celui-ci. Ainsi la priorité inter-DAG pour les tâches est définie de la manière suivante :

$$\forall a_n \in \mathcal{A}, rank_{interDAG}^{G-HEFT}(t_i^{a_n}) = rank_{intraDAG}(t_i^{a_n}) = rank_{HEFT}(t_i^{a_n})$$

Notez qu'il n'est pas nécessaire de construire le DAG virtuel : Il suffit de calculer la priorité de chaque tâche du nouveau DAG et d'insérer les tâches dans la liste triée des tâches à exécuter.

AGING G-HEFT

Un des problèmes engendrés par l'heuristique précédente (G-HEFT) est de mettre en place une liste globale pour toutes les tâches sans distinction de l'application. À chaque DAG soumis, les tâches de ce nouveau DAG sont insérées dans cette liste. Or, la priorité étant basée sur le chemin critique

dans le DAG, les nouvelles tâches arrivées ont potentiellement une priorité supérieure à celles déjà présentes. Elles sont donc insérées en début de liste. Cette heuristique a donc tendance à commencer chaque nouveau DAG, puis elle avance par vagues successives de priorité. Elle finira les DAGs tous en même temps à la fin d'une série de soumissions.

Afin d'atténuer cet effet, nous avons décidé de modifier la priorité inter-DAG en utilisant la priorité HEFT pondérée par une fonction tenant compte de l'âge du DAG dans le système. Ainsi, la priorité d'une tâche devient :

$$\forall a_n \in \mathcal{A}, rank_{interDAG}^{\text{AGING G-HEFT}}(t_i^{a_n}) = rank_{intraDAG}(t_i^{a_n}) * f(\text{âge})$$

Nous avons utilisé deux types de fonctions :

- la priorité intra-DAG est multipliée par un plus l'âge de l'application dans le système divisé par le temps de terminaison estimé (*makespan*). Cette pondération permet de tenir compte de l'âge proportionnellement à l'importance de l'application.

$$f(\text{âge}) = 1 + \frac{\text{âge}}{makespan}$$

- nous prenons l'exponentielle de l'expression précédente, de manière à vraiment privilégier les applications vieilles. Cette heuristique est identifiée par AGING e G-HEFT.

$$f(\text{âge}) = e^{1+\frac{\text{âge}}{makespan}}$$

FCFS

L'heuristique précédente permet de faire passer devant les tâches plus jeunes. Cependant, il n'est pas forcément évident de trouver un juste milieu pour considérer l'âge. Le cas extrême de cette pondération devenant l'heuristique *First Come First Serve*, qui considère les tâches par ordre d'arrivée dans le système. Ainsi, la priorité inter-DAG devient la date d'arrivée de l'application DAG dans le système :

$$\forall a_n \in \mathcal{A}, rank_{interDAG}^{FCFS}(t_i^{a_n}) = r_i^{a_n}$$

Où $r_i^{a_n}$ correspond à la date d'arrivée de l'application dans le système.

SRPT

Cette heuristique attribue au rang inter-DAG la somme des quantités de travail moyen qu'il reste à effectuer dans l'application, c'est à dire :

$$\forall a_n \in \mathcal{A}, rank_{interDAG}^{SRPT}(t_i^{a_n}) = \sum_{t_i \text{non terminée}} \overline{w_i}^{a_n}$$

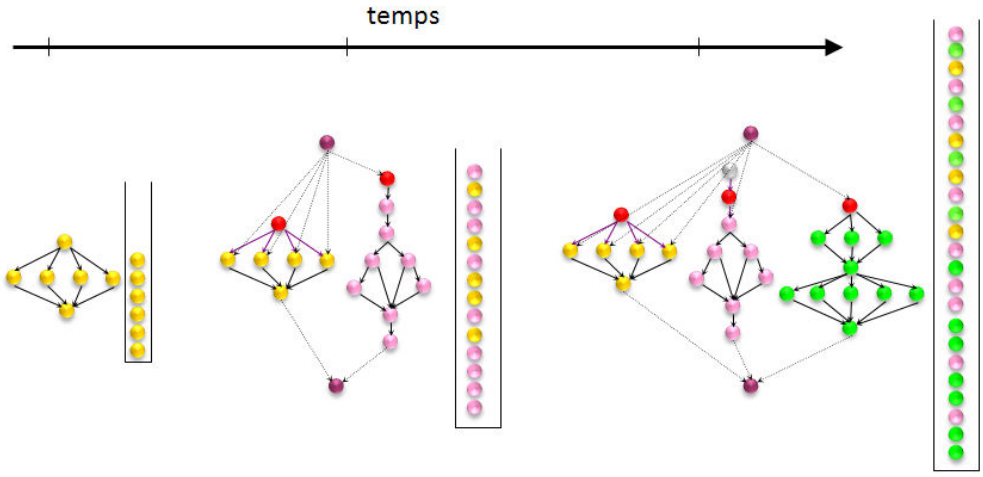


Figure 5.8 – Priorité inter-DAG proportionnelle à celle intra-DAG (G-HEFT, AGING G-HEFT).

Ce qui revient à terminer les applications dont il reste le moins de travail à faire (*Shortest Remaining Processing Time*).

FOFT

Cette dernière heuristique introduit la notion de ralentissement. Nous allons chercher à privilégier l'application qui a subi le plus fort ralentissement. Nous définissons le ralentissement ou *slowdown* par :

$$slowdown = \frac{\text{estimation du temps de fin}}{\text{estimation du temps de fin, si seul dans le système}}$$

Le ralentissement est donc le rapport entre le temps estimé d'exécution actuellement et le temps estimé d'exécution si l'application avait été seule sur la plate-forme. Ce qui revient à privilégier le DAG qui a été le plus pénalisé par le partage de la plate-forme avec les autres applications (*Fairness On Finish Time*).

Les figures 5.8 et 5.9 illustrent le fonctionnement des heuristiques :

- les heuristiques (G-HEFT, AGING G-HEFT) ont une priorité inter-DAG proportionnelle à la priorité intra-DAG. Ce qui revient à maintenir une liste globale de tâches à considérer dès que celles-ci sont prêtes.
- les heuristiques (FOFT, SRPT, FCFS) effectuent d'abord un choix à partir de la priorité inter-DAG puis suivant la priorité intra-DAG. Elles considèrent donc les applications plutôt que les tâches dans leur ensemble.

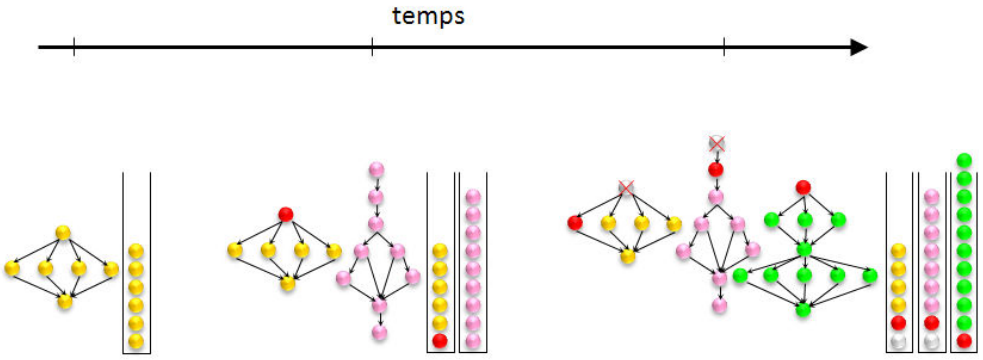


Figure 5.9 – Indépendance des priorités inter-DAG et intra-DAG (FOFT, FCFS, SRPT).

Au final 5 heuristiques ont été présentées, elles explorent plusieurs façons de fixer une priorité inter-application. La priorité intra-application fixée par l'heuristique HEFT est conservée. Une sixième heuristique baptisée BASIC a été créée, elle ne tient pas compte du fait qu'il existe plusieurs applications dans le système. Elle déroule l'heuristique HEFT pour chaque application de manière indépendante.

Nous décrivons par la suite, une architecture permettant l'implantation de ces heuristiques dans l'intergiciel DIET.

5.4.4 Architecture de gestion des graphes de tâches dans DIET

Nous avons présenté dans le chapitre 3 le fonctionnement détaillé de l'intergiciel DIET. Principalement, en tant que NES (*Network Enabled Server*), DIET a été conçu pour déployer un ensemble de services sur une grille de calcul et orchestrer l'ordonnancement des requêtes de clients qui désirent accéder à ces services. Une des spécificités de DIET est de permettre de définir des ordonnanceurs spécifiques adaptés au problème que nous voulons traiter. Il existe par ailleurs une gestion interne des données permettant la persistance de celles-ci [30] sur les serveurs de calcul gérés par DIET. Tous les éléments sont donc réunis pour le développement et l'intégration d'une gestion des *workflows*.

Tous les gestionnaires de *workflows* que nous avons présentés auparavant utilisent, au mieux, une heuristique de liste pour ordonnancer et allouer la tâche sur les ressources qu'ils gèrent, ou au moins un ordonnancement, qui

respecte les contraintes de dépendance, et une allocation décidée par l'utilisateur.

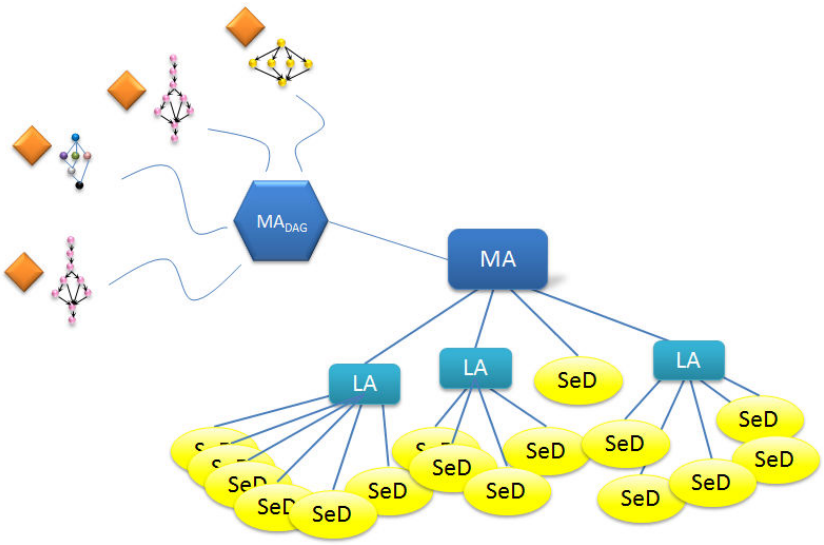


Figure 5.10 – Architecture MA_{DAG}.

Nous avons décidé de mettre en place une architecture logicielle permettant la mise en place d'une heuristique de liste en laissant la possibilité de modifier les différentes étapes (a,b,c) (cf. Algorithme 2) de l'heuristique de liste.

Un agent supplémentaire a été introduit, baptisé MA_{DAG}. Ce composant est responsable de la gestion de l'exécution des graphes de tâches décrits et envoyés par un client. Le MA_{DAG} peut être vu comme un élément externe de la hiérarchie DIET, il n'est pas indispensable à la plate-forme et un client peut aussi directement soumettre des requêtes à DIET sans pour autant avoir recours au MA_{DAG}. Il est d'ailleurs tout à fait envisageable que le client gère lui-même l'exécution d'un ensemble de services sans pour autant solliciter le MA_{DAG}. La figure 5.10 illustre l'architecture de gestion des graphes de tâches dans DIET.

5.4.5 Fonctionnement du MA_{DAG}

La description d'un graphe de tâches dans DIET se fait par l'intermédiaire d'un fichier XML. Il correspond à la description des services appelés et des dépendances de données entre les services.

Nous allons décrire l'exécution d'un graphe de tâches pour un cas particulier.

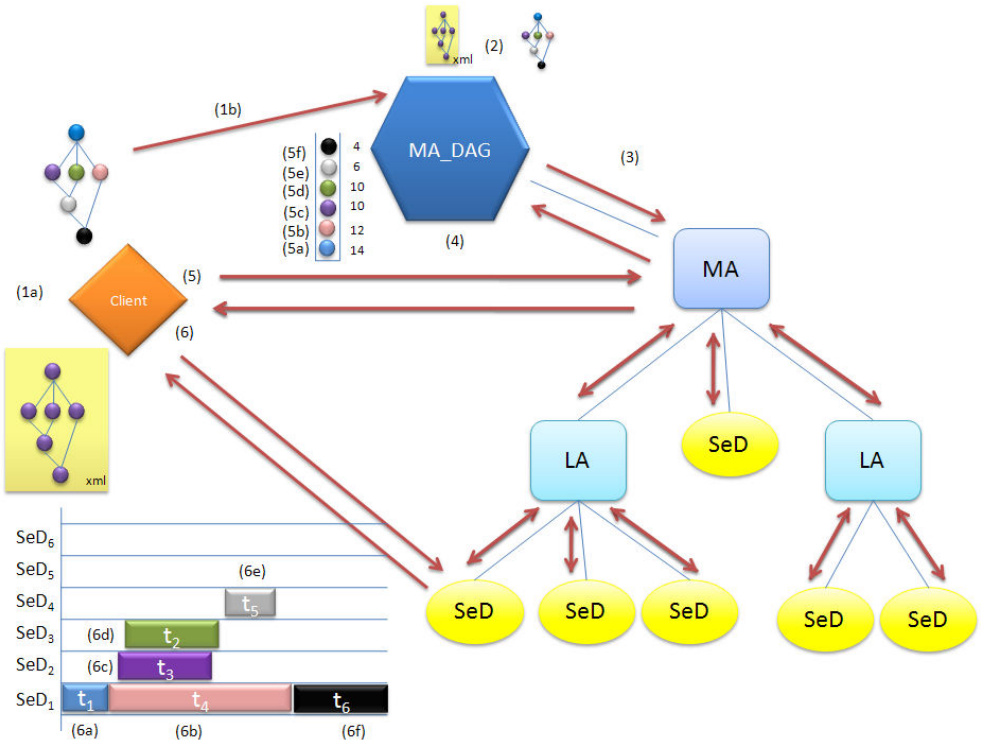


Figure 5.11 – Exemple d'exécution d'un DAG dans DIET.

Les différentes étapes illustrées par la figure 5.11 sont :

étape (1) Le client génère la structure du graphe de tâches localement - étape (1a). Cette structure est constituée de tous les profils (tâches) qui seront appelés lors de la résolution de l'application. Le client se connecte au MA_{DAG} et lui transmet le fichier XML de son DAG - étape (1b).

étape (2) Le MA_{DAG} lit le fichier XML et génère les descriptions de profils (2). La description d'un profil ne contient pas de données. Il n'y a aucun transfert de données qui transite via le MA_{DAG}.

étape (3) le MA_{DAG} calcule ensuite le rang de chaque tâche suivant l'heuristique HEFT. Pour le calculer, l'heuristique utilise la moyenne des temps d'exécution et le temps maximum de transfert de données. L'étape (3) est donc constituée d'une séquence de demandes d'estimation du temps d'exécution pour chaque tâche, en passant par le *Master*

Agent (MA) (étapes 3a,3b,3c,3d,3e,3f). Ce dernier lui retourne la liste de tous les SeDs capables d'exécuter le service et son estimation du temps de calcul pour cette tâche. La liste des SeDs est obtenue grâce au MA qui parcourt la hiérarchie des agents et remonte l'estimation de performance demandée à chaque SeD.

étape (4) À partir du rang de chaque tâche, le MA_{DAG} établit l'ordre d'exécution intra-DAG des tâches. L'exécution du graphe de tâches peut alors commencer - étape (5).

étapes (5) et (6) La première tâche t_1 est mise dans l'état *ready* et le MA_{DAG} recontacte le client pour l'informer qu'il peut exécuter cette tâche. t_1 est mise dans l'état *running* étape (5a). Le client contacte la hiérarchie DIET passant cette fois par le MA directement de manière à obtenir le SeD qui terminera la tâche au plus tôt. Notez la différence entre les étapes (3a) et (5a) où les demandes à la hiérarchie ne sont pas les mêmes : durant l'étape (3a) nous demandons l'estimation du temps d'exécution de la tâche. À l'étape (5a)[9] nous demandons le temps de terminaison au plus tôt de la tâche. La valeur retournée tient compte de la charge actuelle de la machine. Ainsi, si le SeD interrogé a déjà d'autres tâches en cours ou en attente d'exécution, il retournera le temps de fin de la dernière tâche plus le temps d'exécution de la tâche demandée. Ensuite, le client contacte le SeD (obtenu grâce à la hiérarchie DIET) qui terminera au plus tôt la tâche t_1, transfère ses données initiales si nécessaire et invoque le service lié à t_1 (étape 6a). Une fois la tâche t_1 terminée, les données en sortie de t_1 sont déclarées persistantes et stockées sur le serveur qui a exécuté la tâche. Les tâches t_2, t_3 et t_4 passent à l'état *ready*.

De par l'ordre défini par le rang, nous commençons ici par la tâche t_4 et de la même manière que pour la tâche t_1 le client contacte la hiérarchie pour obtenir le SeD qui terminera au plus tôt (étape 5b) en tenant compte de la localité des données. En effet, les données étant stockées dans les SeDs et répertoriées par un identifiant, le temps de terminaison au plus tôt inclura le temps de transfert des données.

Puis la tâche t_3 suivie de t_2 est exécutée (étapes 6c,6d). À ce moment, trois tâches sont exécutées en parallèle par la plate-forme. La tâche t_3 se termine, aucune autre tâche ne passe dans l'état *ready*. La tâche t_3 se termine rendant la tâche t_5 prête et exécutable par le client (étapes 5e,6e). Lorsque les tâches t_5 et t_4 se terminent, t_6 passe à l'état *ready* (5f). La dernière tâche t_6 s'exécute (étape 6f). À la fin de l'exécution

[9] L'étape (5a) est identique à la description que nous avons faite du déroulement d'une requête DIET dans le chapitre précédent.

de toutes les tâches le client récupère les données de son application DAG grâce aux identifiants de celle-ci.

Nous observons, lors du déroulement de l'algorithme, que le rang des tâches est obtenu au début de la soumission du DAG au MA_{DAG}. Puis, au cours de l'exécution nous utilisons les données dynamiques de la plate-forme au moment du choix de la machine qui accomplira la tâche. L'implantation de l'heuristique HEFT proposée ici est donc semi *on-line* (semi à la volée). En aucun cas l'ordre défini par le rang des tâches est un ordre bloquant. On n'attend pas la fin de l'exécution d'une tâche s'il existe une tâche prête, même si celle-ci a une priorité inférieure. Cet ordonnancement respecte le principe d'ordonnancement de liste.

5.4.6 Implantation des heuristiques multi-DAGs

Le fonctionnement précédemment décrit illustre le principe d'ordonnancement à la volée des DAGs défini par l'heuristique HEFT. C'est l'algorithme de base implanté par le MA_{DAG}. Dans ce mode, que nous avons baptisé BASIC, si deux clients contactent le MA_{DAG} de telle sorte que les exécutions se chevauchent, il n'y aura pas de « coopération » ou « arbitrage » mis en place. Chaque client suivra le cours de l'exécution de son application. Cependant, ils se partageront les ressources de la plate-forme. Ainsi, l'allocation produit remplira la file d'exécution (FIFO : *First In First Out*, premier entré premier sorti) de chaque SeD en choisissant la ressource qui terminera au plus tôt la tâche compte tenu de sa charge.

Nous avons implanté les cinq heuristiques *multi-DAG* décrites précédemment au niveau du MA_{DAG}. Celui-ci joue le rôle de point central et permet d'orchestrer la gestion des ressources entre les DAGs qui lui sont soumis. Il arbitre les exécutions concurrentes des *workflows* clientes. L'heuristique *multi-workflows* utilisée par le MA_{DAG} est configurée au lancement en spécifiant l'option sur la ligne de commande. Une fois démarré, il exécutera toujours la même heuristique.

Pour chaque soumission d'un DAG par un client, le MA_{DAG} construit la liste de tâches à partir du fichier xml envoyé par le client et il l'ordonne suivant le rang défini par l'heuristique HEFT. Contrairement à l'implantation du mode BASIC, il bloque les tâches prêtes tant qu'il n'y a pas une ressource libre. Au moment où une ressource se libère (fin d'une tâche ou apparition d'une nouvelle ressource), le MA_{DAG} donne l'ordre au client dont la tâche a été sélectionnée de lancer l'exécution. L'arbitrage inter-DAG s'effectue uniquement lorsqu'il existe plus de tâches prêtes qu'il n'y a de ressources libres. Si cette limite n'est pas atteinte, les 5 heuristiques se comportent toutes exactement

comme le mode BASIC que nous avons décrit précédemment. Au niveau des clients, il n'y a aucun changement quelque soit l'heuristique utilisée.

Nous rappelons que le MA_{DAG} ne gère que des références aux données. Aucun résultat ou paramètre ne transite par son intermédiaire, il orchestre les « tops » envoyés aux clients afin qu'ils déclenchent l'exécution de leurs calculs. De même les données intermédiaires des calculs ne sont pas retournées au client s'il n'en a pas besoin.

5.4.7 Quelques mots sur le passage à l'échelle

La charge de l'ordonnancement est répartie entre le MA_{DAG} et la hiérarchie DIET. Le client est responsable de l'envoi des données et du fichier XML de description du DAG au MA_{DAG}, puis le client est piloté par le MA_{DAG} qui lui donne les ordres d'exécution pour chaque tâche après en avoir établi le rang. Toute la charge de l'allocation des ressources est déléguée à la hiérarchie DIET. Nous avons démontré dans le chapitre 4 la capacité de passage à l'échelle de DIET en montrant qu'une plate-forme DIET était capable de supporter une hiérarchie avec 574 SeDs tout en gardant des temps de réponse pour l'allocation de ressources de l'ordre de la seconde.

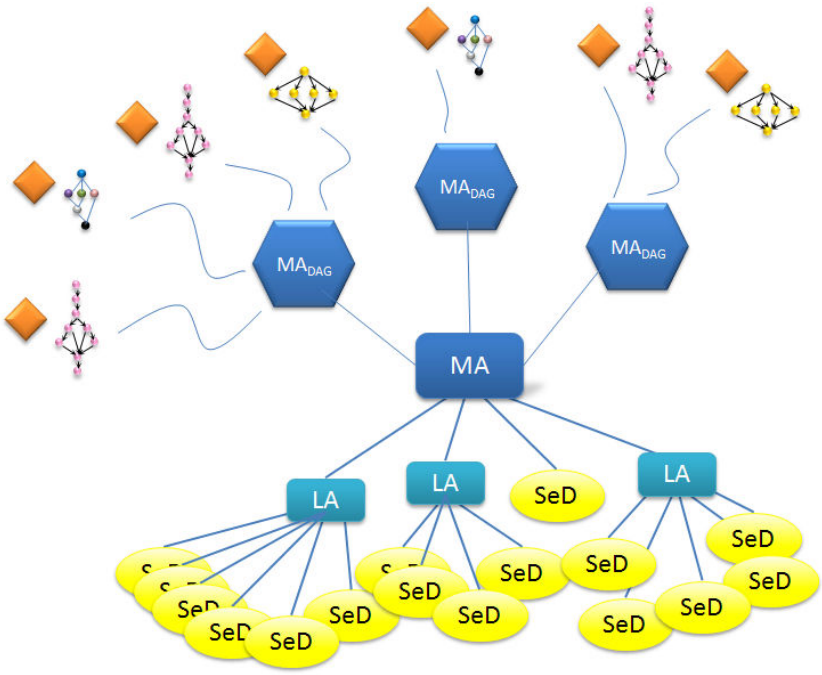


Figure 5.12 – Architecture multi-MA_{DAG}.

Le MA_{DAG} est un point central d'accès pour les clients désirant utiliser les services d'orchestration des applications de type graphes de tâches. L'implantation du MA_{DAG} permet de multiplier le nombre de MA_{DAG} (voir figure 5.12), et nous pouvons en déployer autant que nécessaire. Le déploiement de plusieurs MA_{DAG} rendra inefficace les heuristiques multi-DAG, mais nous pouvons imaginer regrouper les clients faisant accès à un sous ensemble de services. Dans ce cas, chaque MA_{DAG} pourra arbitrer l'affectation des ressources entre les différents clients. Une autre possibilité envisageable est le déploiement de plusieurs hiérarchies DIET sur un ensemble de ressources avec un MA_{DAG} par hiérarchie.

Il faut néanmoins garder à l'esprit que l'arbitrage multi-DAG, ne prend de sens que si le nombre de tâches à exécuter en parallèle devient supérieur au nombre de ressources disponibles. C'est à dire, dans le cas particulier où 574 SeDs sont déployés, qu'il faut soumettre des graphes de tâches ayant un degré de parallélisme de cet ordre de grandeur, ou un nombre concurrent de DAG dont la somme des degrés de parallélisme dépasse le nombre de ressources de la plate-forme DIET.

5.4.8 Expériences et validations

Dans le but de valider les algorithmes et leur implantation dans le MA_{DAG}, nous avons mené une série d'expériences en utilisant 3 applications différentes issues du programme Décrypthon.

L'application MAXDo : la figure 5.14 modélise l'application sous la forme d'un graphe de tâches. Cette application consiste en une première étape où l'on détermine les paramètres d'amarrage à utiliser pour le couple de protéines (ligand et récepteur). Ensuite, ces paramètres sont envoyés vers 4 tâches distinctes qui effectueront les calculs d'amarrages moléculaires et de minimisation d'énergie. La dernière tâche agrège les résultats et les analyse pour donner la carte d'énergie d'interaction entre les deux protéines étudiées et des données statistiques sur les minimisations effectuées aux étapes précédentes.

L'application GEE : cette application de bioinformatique travaille sur des banques protéiques. La figure 5.14 illustre l'enchaînement des étapes sous la forme d'un graphe de tâches. La première tâche prend en entrée les paramètres du client qui sont les espèces étudiées et l'adresse des banques de données. Les sept tâches suivantes filtrent les banques protéiques suivant les sept espèces étudiées. L'étape intermédiaire agrège les données extraites des espèces en une banque spécifique constituée uniquement des informations issues des sept espèces cibles de l'étude.

La dernière série de tâches effectue une recherche de séquences similaires dans la banque d'espèces construite précédemment et les séquences de protéines fournies par l'utilisateur. La dernière étape regroupe les résultats, et établit la classification et l'intégration de l'information d'expression dans une ontologie inter-espèces.

L'application PipeAlign : est une succession d'enchaînements de tâches dans le but de fournir un alignement de protéines de qualité. La figure 5.14 illustre l'enchaînement des tâches impliquées dans pipeAlign. La première étape extrait les séquences similaires dans une banque de données. Ces données sont ensuite traitées (*ballast*) et filtrées (*filtering*). Un premier alignement est produit (*clustalw*) et raffiné par la suite par deux autres programmes produisant eux aussi un alignement (*rascal*, *leon*). À la suite de chaque alignement produit, celui-ci est évalué par le programme (*normd*).

Ces trois applications ont été portées dans l'intergiciel DIET. Chaque programme constituant les tâches du *workflow* scientifique est intégré dans un SeD de façon à pouvoir faire appel à ce service individuellement. Ainsi, un SeD est capable d'accomplir les 3 services de l'application MAXDo, les 4 services de l'application GEE et les 7 services de pipeAlign, soit 14 services en tout.

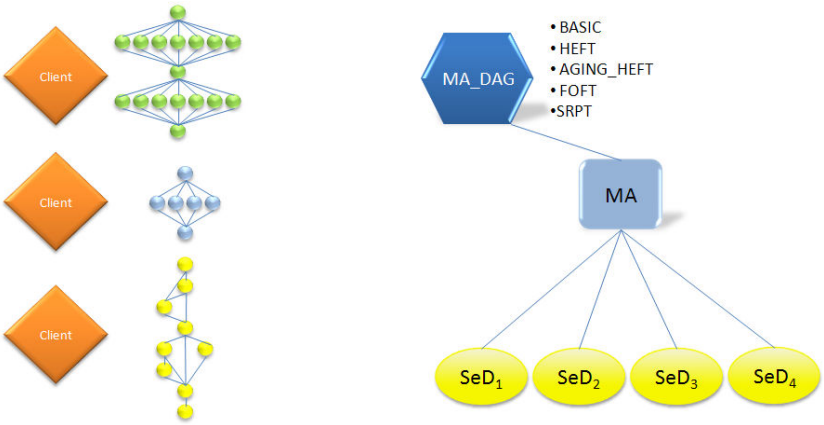


Figure 5.13 – Plate-forme de test.

La plate-forme employée est illustrée par la figure 5.13. Les SeDs sont déployés sur une machine distincte, le *MasterAgent* et le MA_{DAG} sont déployés sur la même machine. Nous utilisons une machine différente pour les clients. Chaque SeD n'exécute qu'une seule tâche à la fois, et si plusieurs requêtes

	MAXDo	PipeAlign	GEE
makespan	33 s	1 min 36 s	2 min 3 s
$\sum w$	1 min 20 s	2 min 5 s	3 min 20 s

Tableau 5.2 – Makespans pour chaque application seule sur la plate-forme (moyenne sur 3 exécutions).

lui sont attribuées, il les traite dans l'ordre d'arrivée (FIFO[10]). Pour chaque service, nous sommes en mesure de donner une évaluation précise des temps de calcul nécessaires et du volume de données échangées. Par commodité, le temps de calcul comprend le temps de transfert des données, le réseau connectant les machines étant ici homogène.

Nous explicitons le comportement des heuristiques dans une configuration particulière pour un scénario d'expérience défini. La figure 5.14 montre les graphes de tâches applicatifs utilisés pour cette expérience. Ces graphes comportent la valuation des temps de calcul moyens inscrits dans le sommet des graphes et la valeur du chemin critique (i.e la priorité donnée par l'heuristique HEFT) pour chacune des tâches.

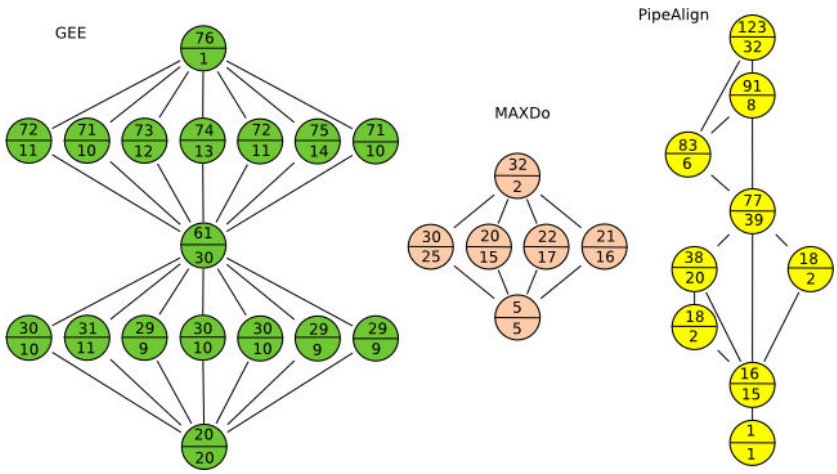


Figure 5.14 – DAG valué des 3 applications.

Le tableau 5.2 fournit les temps d'exécution (*makespan*) de chaque application seule sur la plate-forme. Il est inutile de comparer les temps d'exécution pour les différentes heuristiques car elles ont exactement le même comporte-

[10] *First In First Out* : premier entré premier sorti

ment lorsqu'il n'y a pas de soumissions concurrentes. Nous avons donc ici le temps d'exécution obtenu avec l'heuristique HEFT.

L'expérience menée suit le scénario suivant :

- À la date r_0, début de l'expérience, le premier client demande l'exécution d'un graphe de tâches. Ensuite, toutes les 12 s, un nouveau client demande une nouvelle exécution d'une application sélectionnée au hasard parmi l'ensemble des applications qui n'ont pas encore été soumises. La dernière soumission de cette première série se termine à r_0 + 600 s. La première série comporte 10 appels à l'application GEE, 25 appels à l'application MAXDo et 16 appels à l'application PipeAlign.
- À la date $r_0 + 1200$ s, nous recommençons une autre série aléatoire avec le même nombre d'appels à chaque application. La dernière soumission s'effectue à $r_0 + 1800$ s.

Au total, nous avons 102 soumissions de DAGs différents échelonnées toutes les douze secondes en deux séries de 600 s sur une plate-forme formée de 4 SeDs. Pour cette expérience nous avons choisi des paramètres en entrée de chaque application de façon à avoir une quantité de calcul demandée comparable. C'est à dire : $\sum w^{GEE} = \sum w^{MAXDo} = \sum w^{PipeAlign}$.

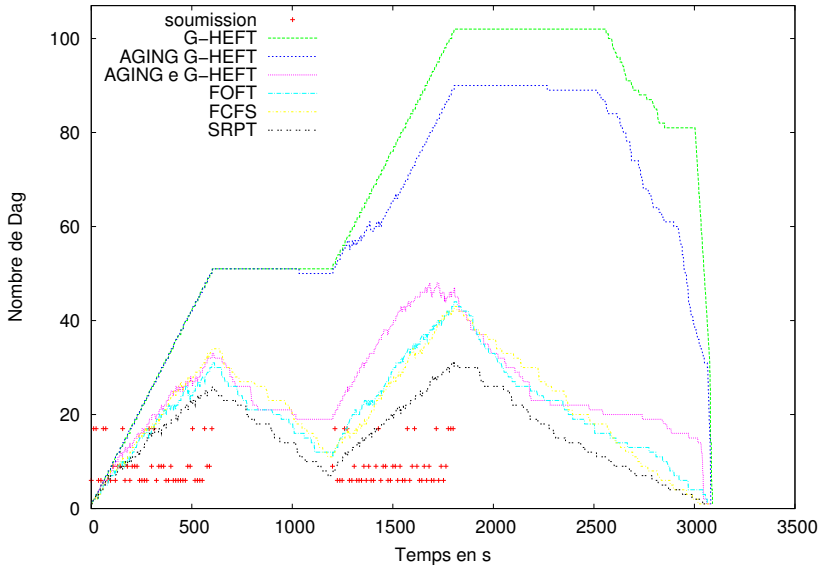


Figure 5.15 – Nombre de DAGs présents dans le système durant l'expérience.

La figure 5.15 trace la courbe du nombre de DAGs encore en exécution dans le MA_{DAG} du début jusqu'à la fin de l'expérience pour chaque heuristique. Les croix rouge représentent les temps de soumission d'un nouveau

DAG dans le système, leur hauteur symbolise le nombre de tâches dans le DAG. Ainsi, un point à une hauteur de dix-sept représente l'application GEE, neuf pour PipeAlign et six pour MAXDo.

À la vue de ces courbes (figure 5.15) plusieurs remarques sont à faire :

- Pour toutes les heuristiques, le temps total de terminaison de l'expérience est similaire. En particulier, si l'on compare ce temps à une borne inférieure du temps total d'exécution réparti sur quatre processeurs, sans tenir compte des dépendances du graphe de tâches et de l'indivisibilité des tâches, alors les heuristiques sont entre 2% et 3,5 % de cette borne inférieure inatteignable.
- Les heuristiques qui prennent en compte une priorité inter-DAG (FOFT, FCFS, SRPT) et AGING e G-HEFT[11] finissent les DAGs plus tôt.
- À l'inverse, l'heuristique G-HEFT, AGING G-HEFT terminent les DAGs quasiment en même temps à la fin de l'expérience.

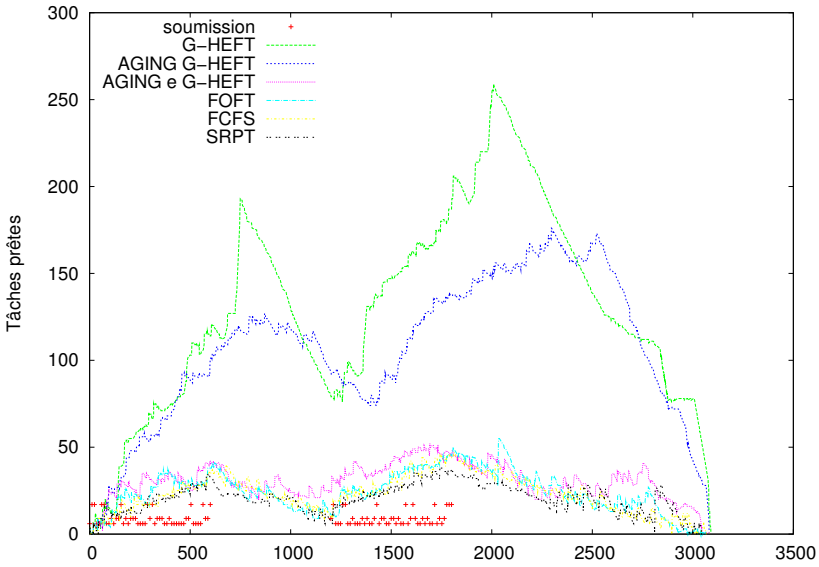
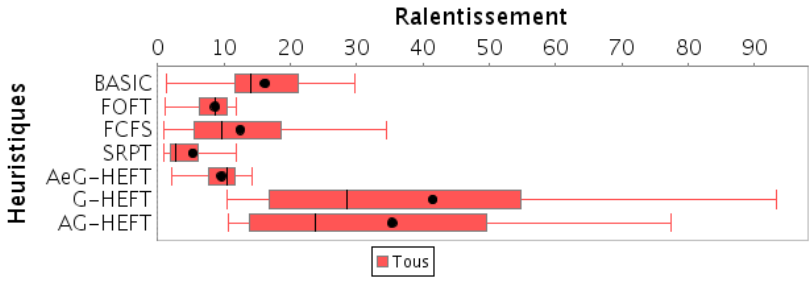


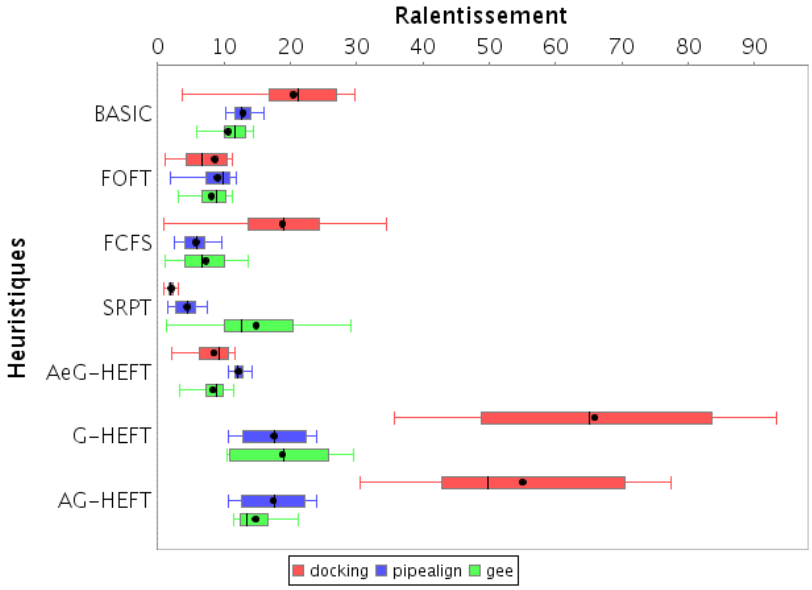
Figure 5.16 – Nombre de tâches prêtes à être exécutées.

La figure 5.16 illustre la progression par étage de l'heuristique G-HEFT. En effet la priorité inter-DAG est égale à la priorité intra-DAG. Comme tous les DAGs d'un même type sont identiques, cette heuristique a tendance à progresser par vagues. Le fait d'introduire une pondération sur la priorité inter-DAG (heuristique AGING G-HEFT) permet de lisser ce comportement,

[11]La fonction utilisée dans l'heuristique AGING G-HEFT est la fonction exponentielle.



(a) Ralentissement pour toutes les applications sans distinction ;



(b) Ralentissement pour chaque application (MAXDo, pipeAlign, GEE) ;

Figure 5.17 – Boîtes à moustaches des ralentissements suivant les heuristiques.

nous ne retrouvons pas ce phénomène dans les autres heuristiques (FCFS, FOFT, SRPT, AGING e G-HEFT).

Nous présentons aussi sous la forme d'un graphique dit « boîte à moustache[12] » [112], les données statistiques des ralentissements subits par les applications. La longueur de la boîte correspond à la longueur de l'intervalle inter-quartile [13] ($Q_3 - Q_1$), le trait et le point à l'intérieur de la boîte marquent respectivement la médiane et la moyenne des valeurs, les deux « moustaches » sortant des boîtes symbolisent le ralentissement minimum et maximum. Le

[12]Traduction de *Box & Whiskers Plot*

[13]Quartile : chacune des quatre parties, d'effectif égal, d'un ensemble ordonné de valeurs.

ralentissement est défini par le temps d'exécution de l'application lors de l'expérience divisée par le temps d'exécution de l'application si elle avait été seule à s'exécuter sur la plate-forme (cf. tableau 5.2). Plus la valeur est proche de 1, moins l'application est ralentie.

La figure 5.17 montre deux graphiques différents représentant respectivement le ralentissement pour toutes les applications sans distinction (voir figure 5.17(a)) et le ralentissement séparé pour chacune d'elles (voir figure 5.17(b)).

Sans surprises, l'heuristique G-HEFT obtient le plus fort ralentissement moyen. Dans l'absolu cette heuristique pourrait ne jamais finir aucun DAG s'il y avait un flux continu de soumission. Nous observons que le fait de pondérer la priorité inter-DAG avec une valeur qui dépend de la vieillesse de l'application dans le système permet une diminution du ralentissement (comparaison entre G-HEFT et AGING G-HEFT). En outre, dans le scénario choisi, l'heuristique SRPT obtient le plus faible ralentissement moyen suivie des heuristiques FOFT, AGING e G-HEFT et FCFS. En effet, le critère de choix SRPT favorise l'application MAXDo qui demande le moins de travail et qui est soumise le plus grand nombre de fois (50 fois en tout). De plus, l'application MAXDo possède le plus faible *makespan* (voir tableau 5.2), elle est donc le plus sensible au ralentissement. En particulier, nous observons dans la figure 5.17(b) que l'heuristique FOFT tend à homogénéiser les ralentissements des trois applications, contrairement à SRPT qui avantage les applications courtes et FCFS qui a tendance à pénaliser l'applications MAXDo au profit des autres. Enfin, l'heuristique AGING e G-HEFT, en définissant une priorité inter-DAG qui dépend de la priorité intra-DAG de l'algorithme HEFT et qui dépend fortement (exponentiel) du rapport entre l'âge et le *makespan* du DAG, conjugue le principe de FCFS, FOFT et de SRPT (voir les courbes de la figure 5.15). En effet, les applications avec un faible *makespan* voient le rapport âge/*makespan* augmenter plus rapidement que les autres. De plus, la considération de l'exponentiel de l'âge d'une tâche rapproche l'heuristique du comportement de FCFS

Intéressons nous à l'équité de ces heuristiques. Il existe des manières différentes de définir l'équité [15, 51, 119], cette mesure jauge les différences de traitement entre les applications par rapport à une métrique. Nous définissons cette mesure de dispersion par rapport au ralentissement (*slowdown*) que subit chaque application. Plusieurs valeurs expriment l'étalement et donc l'équité :

- L'étendue : différence entre la valeur maximum et la valeur minimum, cette mesure est très sensible aux valeurs extrêmes ;
- L'écart interquartile : différence entre le premier et le troisième quartile ;
- L'écart moyen : moyenne arithmétique des écarts absolus par rapport

à la moyenne ;
– L'écart type : moyenne quadratique des écarts par rapport à la moyenne.

La représentation en « boîte à moustache » des valeurs du ralentissement permet de rendre compte de l'étalement des résultats (voir figure 5.17). Nous remarquons que les heuristiques AGING e G-HEFT, FOFT possèdent un étalement des ralentissements faible. Ce qui traduit l'équitabilité de ces heuristiques. Elles obtiennent, de plus, des ralentissements moyens meilleurs que l'heuristique BASIC.

En conclusion, sur ce scénario d'expériences, il apparaît que le temps total d'exécution reste comparable pour toutes les heuristiques. Il est proche d'une borne théorique inférieure (entre 2% et 3,5%). Il apparaît ici que la meilleure heuristique permettant d'obtenir un ralentissement moyen le plus bas est l'heuristique SRPT. Pour ce scénario, nous l'expliquons par le fait que le nombre de petites applications est élevé. Il est donc judicieux de privilégier les petites applications afin d'obtenir un faible ralentissement, d'autant que ce sont justement ces petites applications qui sont le plus sensibles au ralentissement. En ce qui concerne l'équité, ce sont les heuristiques AGING e G-HEFT et FOFT qui obtiennent un étalement des ralentissements faibles tout en gardant une valeur moyenne proche de celle obtenue avec SRPT.

Autres expériences

Nous avons lancé une multitude d'autres expériences permettant de vérifier le comportement des heuristiques. Nous avons, par exemple, cherché à faire varier l'ordre d'arrivée, le nombre d'applications, la taille et l'hétérogénéité de la plate-forme, tout en gardant une quantité de travail constante pour chaque application. Les conclusions apportées pour le scénario que nous avons choisi de commenter restent valables pour les autres expériences que nous avons menées. Néanmoins, il est clair qu'il existe des contre-exemples ou des situations qui mettront en défaut une ou plusieurs des heuristiques. De manière générale, il apparaît que l'heuristique G-HEFT est à proscrire pour l'ordonnancement de plusieurs graphes de tâches lorsque nous observons le ralentissement subi par les applications. En revanche, elle fournit des performances efficaces lorsqu'il s'agit de l'ordonnancement intra-DAG. Les heuristiques FOFT et AGING e G-HEFT sont les heuristiques qui permettront d'être le plus équitable entre les applications lorsque celles-ci entrent en concurrence pour les ressources. Et suivant les proportions des différentes applications il peut être opportun de mettre en place l'heuristique SRPT.

5.5 Conclusion

Au cours de ce chapitre, nous avons présenté de manière générale les problèmes d'ordonnancement de graphes de tâches sur les grilles de calcul. Nous nous sommes intéressés au problème particulier de l'ordonnancement à la volée de plusieurs graphes de tâches dans un environnement hétérogène. Après avoir établi un tour d'horizon des différentes techniques d'ordonnancement disponibles pour l'ordonnancement mono-DAG, nous avons proposé six heuristiques différentes permettant l'ordonnancement de plusieurs DAGs soumis par des utilisateurs différents. Elles sont basées sur l'heuristique de liste HEFT [111] que nous avons adaptée pour être utilisable dans un contexte plus dynamique.

De plus, nous avons développé une architecture logicielle dans l'intergiciel DIET permettant la mise en place d'une heuristique de liste, et plus particulièrement HEFT qui a été montrée comme une heuristique efficace dans de nombreux cas. Pour cela, nous avons ajouté un nouveau composant logiciel : le MA_{DAG}. Il prend en charge l'exécution du graphe de tâches décrit par le client sous la forme d'un fichier XML. L'architecture proposée permet en outre un passage à l'échelle aisé, une indépendance et une robustesse vis à vis de l'intergiciel DIET. Cette implantation n'est pas intrusive dans l'architecture DIET et n'a pas demandé de modification particulière dans le fonctionnement général de l'intergiciel. Nous avons ensuite étendu cette architecture pour permettre l'implantation des heuristiques multi-DAGs que nous avons proposées. Elles permettent l'ordonnancement concurrent de plusieurs DAGs en s'appuyant sur les atouts et spécificités de l'intergiciel DIET.

En outre, nous avons validé l'implantation réalisée en utilisant 3 applications issues du Décrypthon modélisées sous la forme de graphes de tâches. Il apparaît, à la vue des tests, que les heuristiques multi-DAG proposées présentent toutes de bonnes performances absolues en terme de temps d'exécution total et d'utilisation des ressources. En revanche il apparaît de façon marquée qu'il ne faut pas utiliser l'heuristique G-HEFT lorsqu'on se trouve dans un contexte dynamique de soumission de nouveaux DAGs. L'exploration d'une version AGING G-HEFT prenant en compte la vieillesse des DAGs dans le système permet d'améliorer les ralentissements subis par les applications. Nous avons pu mettre en évidence, le comportement équitable des heuristiques FOFT et AGING e G-HEFT. Toutefois, dans un contexte où il existe plus de petites applications, il est judicieux d'utiliser l'heuristique SRPT qui privilégie celles-ci.

Chapitre 6

Conclusions et Perspectives

Le domaine des grilles informatiques est un thème de recherche qui connaît un engouement croissant depuis quelques années. Plusieurs raisons peuvent être avancées :

- la mise en commun des ressources permet de repousser les frontières des calculs réalisables dans une échelle de temps humaine et entretient l'illusion d'une puissance infinie. Elle permet aussi d'étendre les capacités de stockage ;
- l'utilisation des ressources des particuliers pour le calcul scientifique rapproche le grand public des recherches intimistes des scientifiques. Ce qui permet aussi de lui donner l'impression de contribuer à l'avancée des connaissances ;
- les problèmes techniques et scientifiques engendrés par l'utilisation d'une grille informatique soulèvent de nouveaux défis motivants pour les ingénieurs et les chercheurs, les obligeant à revoir parfois leur approche scientifique.
- la rencontre de domaines scientifiques aux thématiques différentes sur un même support, la « grille informatique », favorise les nouvelles collaborations et le brassage des idées. Cette émulation entretient une recherche communautaire, pluridisciplinaire, et sans frontières.

Nous avons présenté le programme Décrypthon et la mission d'accompagnateur de projets bioinformatiques vers les technologies de grilles qu'il s'est fixé. Cette initiative démontre une nouvelle fois que les grilles informatiques apportent une réponse tangible aux problématiques de certains projets scientifiques. Nous avons exposé notre travail d'architecte de la grille connectant les ressources de six centres de calcul universitaires. Dans ce travail nous avons permis à des projets de mener à bien leurs calculs sur la grille

Décrypthon en leur offrant une transparence d'utilisation et un support à l'adaptation de leurs programmes. Cet effort se matérialise par l'adoption depuis un an de l'intergiciel DIET, développé dans l'équipe GRAAL, comme intergiciel de production de la grille Décrypthon et par la mise à disposition du DIET_Webboard. Ce portail web développée pour la grille Décrypthon sera à terme librement utilisable pour orchestrer n'importe quelle grille informatique supportée par l'intergiciel DIET.

Il apparaît, dans notre travail, que le processus de « gridification » d'une application existante nécessite encore une étape d'adaptation qui n'est toujours pas transparente. Nous ne sommes pas très loin de la métaphore des grilles avec le réseau électrique proposé par Ian Foster et *al.* Il reste encore le processus de fabrication de la « prise à la grille » qui n'est encore réalisable qu'avec de solides connaissances techniques de l'intergiciel et de l'infrastructure sous-jacente. Dans ce sens, il faut souligner les efforts de standardisation réalisés par les membres de l'OGF. Cependant, même si certains standards existent (JDL, GridRPC, *etc*), il faudra encore du temps avant que ceux-ci soient propagés dans l'ensemble de la communauté informatique.

À travers les expériences que nous avons menées grâce à l'outil de recherche Grid'5000, nous avons contribué à la démonstration de l'intérêt de cet instrument comme environnement de test et de validation. Il apparaît aujourd'hui indispensable de disposer d'un tel instrument pour délimiter les fonctionnalités d'un intergiciel comme DIET qui aspire à la gestion d'un grand nombre de machines. De plus, cet outil qui est véritablement à grande échelle (plus de 1600 machines totalisant près de 4600 cœurs sur 9 sites) nous a permis de démontrer les performances et la capacité d'utilisation d'une hiérarchie DIET sur une telle infrastructure. Nous avons ainsi participé à renforcer l'image de DIET comme intergiciel de type *Network Enabled Server* (NES), modulaire, robuste et léger, capable d'adresser des plates-formes à grande échelle.

Le travail préparatoire accompli sur l'application MAXDo du projet HCMD a contribué à rendre possible l'exécution, sur une grille de volontaires, d'une quantité impressionnante de calcul (plus de 80 siècles de temps processeur en 5 mois). En outre, nous avons introduit la notion de *processeur virtuel à plein temps* (P_{vfp}) qui permet de rendre compte de la dimension d'une grille de volontaires. En 2008, la grille World Community Grid est l'équivalent d'une grille comportant en moyenne plus de 75 000 P_{vfp}. D'autre part, nous nous sommes servis de cette notion pour établir un point de comparaison entre une grille de ressources dédiées et une grille de machines volatiles. Ce facteur de comparaison permet d'établir une projection sur les besoins qu'il sera nécessaire de mettre en place pour la deuxième phase de calcul du projet HCMD. En somme, cette étude complète démontre qu'une

utilisation complémentaire des différentes grilles (Décrypthon, Grid'5000 et World Community Grid) permet de mener à terme une campagne de calcul scientifique.

Enfin, nous avons proposé une architecture modulaire autour de la gestion des tâches ayant des dépendances au sein de l'environnement DIET. L'architecture proposée tire parti des fonctionnalités de l'environnement client-agent-serveur implantées dans l'intergiciel DIET. Après une revue des algorithmes disponibles et des développements analogues, nous avons implanté dans un nouvel élément MA_{DAG} l'heuristique de liste HEFT adaptée aux contextes du modèle DIET. De plus, nous avons proposé un ensemble de six algorithmes permettant l'ordonnancement de plusieurs applications demandant un accès concurrent aux mêmes ressources. Il apparaît dans les résultats de nos tests sur trois applications issues du programme Décrypthon, que les heuristiques ont de bonnes performances d'un point de vue occupation des ressources et temps de terminaison de l'expérience. En revanche il existe de fortes disparités si l'on observe le ralentissement subi par chaque application. Tous les algorithmes que nous avons développés sont disponibles dans la version 2.3 de DIET, librement téléchargeable sur le site `http://graal.ens-lyon.fr/DIET`.

6.1 perspectives

Décrypthon

Dans un premier temps, il sera intéressant de poursuivre le travail préparatoire pour la deuxième phase de calcul du projet HCMD. À cette occasion, nous pourrons infirmer ou confirmer les suppositions que nous avons émises lors de l'évaluation des besoins pour la nouvelle phase de calcul. Nous continuerons le parallèle entrepris pour comparer ces deux types de grilles. Il nous semble important d'observer dans quelle mesure le facteur de comparaison que nous avons établi évoluera avec le renouvellement des machines des volontaires. Nous pensons déjà à baser notre comparaison sur les points accordés aux volontaires lorsqu'ils contribuent aux projets du World Community Grid. De plus, dans un contexte où la préoccupation d'économie d'énergie devient de plus en plus importante pour les particuliers, nous pourront établir les profils de consommation électrique [61] des machines lorsqu'elles travaillent sur les calculs scientifiques. Ainsi, nous serions capables de répondre à la question suivante : quelle est la proportion d'énergie consommée par une machine restant allumée mais inactive par rapport à une machine calculant sur des projets scientifiques. De plus, il serait

intéressant d'établir d'autres points de comparaison avec d'autres travaux semblables menés sur des grilles de production telles que les *data challenge* de l'initiative WISDOM [109] sur la grille européenne EGEE. De plus, dans un avenir proche, les projets du programme Décrypthon pourraient bénéficier d'une incorporation du MA_{DAG} au sein de la grille universitaire Décrypthon. Nous pourrions rendre disponible au sein du DIET_Webboard la gestion d'applications décrites sous la forme d'enchaînement de tâches. Les utilisateurs pourraient alors composer, à la manière de Taverna [100], les programmes « gridifiés » dans la plate-forme universitaire Décrypthon et l'ordonnancement des *workflows* serait orchestré par les heuristiques que nous avons développées et validées.

Gestion des *workflows*

La gestion des *workflows* dans l'intergiciel DIET se fait par la description d'un graphe dirigé orienté (DAG) dans un fichier XML. Or, comme nous l'avons vu dans les approches d'autres moteurs d'exécution de *workflows* (Pegasus/DagMan [28, 92], Triana [67], MOTEUR [41]), l'enchaînement des tâches (*workflow fonctionnel*) est séparé de la description des données. Nous travaillons actuellement au développement d'un langage et d'outils permettant de prendre en compte séparément le *workflow fonctionnel* et les données appliquées à celui-ci.

En outre, il apparaît que notre réflexion et notre travail sur le développement de la gestion de plusieurs applications de type graphes de tâches dans l'intergiciel DIET trouve des échos dans d'autres moteurs d'exécution de *workflows*. Ainsi le projet *Pegasus* de l'*Information Sciences Institute* à l'Université de Californie du Sud développe un outil nommé *Ensemble Manager* [99] permettant de prendre en charge, comme le MA_{DAG}, l'exécution concurrente de *workflows* scientifiques. Nous aurons à cœur d'adapter et d'implanter nos heuristiques au sein de leur environnement.

Générales

Enfin, notre travail autour du programme Décrypthon démontre qu'il est nécessaire de disséminer les connaissances sur les technologies de grilles auprès des scientifiques non spécialistes. Cette étape est très enrichissante humainement et intellectuellement. Elle permet d'appréhender les points de vue et le vocable des domaines scientifiques qui appréhendent l'informatique comme un outil de travail sans chercher forcément à optimiser leurs programmes pour des traitements parallèles. Le rapprochement entre

l'informatique du parallélisme des grilles et les autres sciences explore de nouveaux champs de connaissances pluridisciplinaires.

La diffusion du savoir autour des grilles informatiques demande une maîtrise et une connaissance vaste des différents champs d'action de l'informatique : système d'exploitation, réseaux de communication, programmation, compilation, *etc.* Les grilles informatiques, bien qu'ayant dix ans, si l'on situe la naissance à la parution du livre de Ian Foster et *al* en 1998, ont encore besoin de temps pour arriver à la maturité nécessaire pour être utilisées par tous. Cette diffusion du savoir passera par une rationalisation des technologies (processus en marche avec les groupes de réflexion et de standardisation de l'OGF) et par un mélange des sciences et des ressources au sein de projets motivant comme le Décrypthon, EGEE ou encore le World Community Grid.

Dans cette même perspective, le « cloud computing » reprend les idées des grilles informatiques utilisées par les scientifiques pour en fournir une offre commerciale à destination des entreprises. Un ensemble de puissance de calcul et de mémoire, proposé comme un service à des clients par une entreprise externe. Les entreprises n'auraient ainsi plus besoin de ressources propres, mais confiraient le soin à un prestataire de garantir une puissance de calcul et de stockage à la demande. Dans ce domaine, les mastodontes du monde informatique (Amazon, Google, HP, IBM, Microsoft, Sun Microsystems, *etc*) ont déjà pris place et travaillent à fournir une offre commerciale basée sur la virtualisation répondant aux contraintes de confidentialités et de qualités de services.

Bibliographie

[1] I. Ahmad and Y.-K. Kwok. A new approach to scheduling parallel programs using task duplication. In *ICPP '94 : Proceedings of the 1994 International Conference on Parallel Processing*, pages 47–51, Washington, DC, USA, 1994. IEEE Computer Society.

[2] I. Ahmad and Y.-K. Kwok. On exploiting task duplication in parallel program scheduling. *IEEE Trans. Parallel Distrib. Syst.*, 9(9) :872–892, 1998.

[3] The Globus Alliance. Globus toolkit :an open source software toolkit used for building grids. http ://www.globus.org/toolkit/.

[4] A. Amar, R. Bolze, Y. Caniou, E. Caron, B. Depardon, J.-S. Gay, G. Le Mahec, and D. Loureiro. Tunable scheduling in a GridRPC framework. *Concurrency & Computation : Practice & Experience*, 2008. To appear.

[5] P. Amestoy, M. Daydé, C. Hamerling, M. Pantel, and C. Puglisi. Management of services based on a semantic description within the gridtlse project. In *VECPAR'06 - Workshop on Computational Grids and Clusters (WCGC), Rio de Janeiro, Brésil*, number 4395 in LNCS, pages 634–643, http ://www.springerlink.com/, juillet 2006. Springer-Verlag.

[6] K. Amin, G. Von Laszewski, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi. Gridant : A client-controllable grid workflow system. In *HICSS '04 : Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 7*, page 70210.3, Washington, DC, USA, 2004. IEEE Computer Society.

[7] D.P. Anderson. Boinc : A system for public-resource computing and storage. In *GRID '04 : Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[8] D.P. Anderson, E. Korpela, and R. Walton. High-performance task distribution for volunteer computing. In *E-SCIENCE '05 : Proceedings of the First International Conference on e-Science and Grid Computing*, pages 196–203, Washington, DC, USA, 2005. IEEE Computer Society.

[9] G. Antoniu and M. Bougé, L.and Jan. Juxmem : An adaptive supportive platform for data sharing on the grid. *Scalable Computing : Practice and Experience*, 6(3) :45–55, September 2005.

[10] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Sagi, Z. Shi, and S. Vadhiyar. Users' Guide to NetSolve V1.4. Computer Science Dept. Technical Report CS-01-467, University of Tennessee, Knoxville, TN, July 2001. `http://www.cs.utk.edu/netsolve/`.

[11] R. Bajaj and D. P. Agrawal. Improving scheduling of tasks in a heterogeneous environment. *IEEE Trans. Parallel Distrib. Syst.*, 15(2) :107–118, 2004.

[12] O. Beaumont, V. Boudet, and Y. Robert. The iso-level scheduling heuristic for heterogeneous processors. In *PDP'2002, 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*. IEEE Computer Society Press, 2002.

[13] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I N. Shindyalov, and P. E. Bourne. The Protein Data Bank. *Nucl. Acids Res.*, 28(1) :235–242, 2000.

[14] R. Bolze, E. Caron, F. Desprez, G. Hoesch, and C. Pontvieux. A monitoring and visualization tool and its application for a network enabled server platform. In M. Gavrilova, editor, *Computational Science and Its Applications - ICCSA 2006*, volume 3984 of *LNCS*, pages 202–213, Glasgow, UK., May 8-11 2006. Springer.

[15] T. Bonald, L. Massoulié, A. Proutière, and J. Virtamo. A queueing analysis of max-min fairness, proportional fairness and balanced fairness. *Queueing Syst. Theory Appl.*, 53(1-2) :65–84, 2006.

[16] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *J. Parallel Distrib. Comput.*, 61(6) :810–837, 2001.

[17] E. N. Brown, R. E. Kass, and P. P. Mitra. Multiple neural spike train data analysis : State-of-the-art and future challenges. *Nature Neuroscience*, 7(5) :456–461, May 2004.

[18] P. Brucker. *Scheduling Algorithms (5th edition)*. SpringerVerlag, 2007.

[19] L.-C. Canon, E. Jeannot, R. Sakellariou, and W. Zheng. Comparative evaluation of the robustness of dag scheduling heuristics. Technical Report TR-0120, Institute on Resource Management and Scheduling, CoreGRID - Network of Excellence, December 2007.

[20] J. Cao, A. T. Chan, Y. Sun, S. K. Das, and M. Guo. A taxonomy of application scheduling tools for high performance cluster computing. *Cluster Computing*, 9(3) :355–371, 2006.

[21] J. Cao, S. A. Jarvis, S. Saini, D. J. Kerbyson, and G. R. Nudd. Arms : an agent-based resource management system for grid computing. *Sci. Program.*, 10(2) :135–148, 2002.

[22] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd. Gridflow : Workflow management for grid computing. In *CCGRID '03 : Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, page 198, Washington, DC, USA, 2003. IEEE Computer Society.

[23] E. Caron, A. Chis, F. Desprez, and A. Su. Design of plug-in schedulers for a gridrpc environment. *Future Generation Computer Systems*, 24(1) :46–57, January 2008.

[24] E. Caron, P. Kaur Chouhan, and H. Dail. Godiet : A deployment tool for distributed middleware on grid'5000. In IEEE, editor, *EXPGRID workshop. Experimental Grid Testbeds for the Assessment of Large-Scale Distributed Apllications and Tools. In conjunction with HPDC-15.*, pages 1–8, Paris, France, June 19th 2006.

[25] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Softw. Eng.*, 14(2) :141–154, 1988.

[26] R.W. Conway, W.L. Maxwell, and L.W. Miller. *The Theory of Scheduling*. Addison-Wesley, 1967.

[27] A. Cooke, A. Gray, L. Ma, W. Nutt, J. Magowan, M. Oevers, P. Taylor, R. Byrom, L. Field, S. Hicks, J. Leake, M. Soni, A. Wilson, R. Cordenonsi, L. Cornwall, A. Djaoui, S. Fisher, N. Podhorszki, B. Coghlan, S. Kenny, and D. O'Callaghan. R-gma : An information integration system for grid monitoring. In *Proc.Int.Conf. Cooperative Information Systems (CoopIS'03)*, Catania, Sicily, November 2003.

[28] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus : A framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.*, 13(3) :219–237, 2005.

[29] B. Del Fabbro. *Contribution à la gestion des données dans les grilles de calcul à la demande : de la conception à la normalisation.* Phd thesis, Université de Franche Comté, 2005.

[30] D. Del-Fabbro, B.and Laiymani, J.-M. Nicod, and L. Philippe. Dtm : a service for managing data persistency and data replication in network-enabled server environments. *Concurrency and Computation : Practice and Experience*, 19(16) :2125–2140, November 2007.

[31] Alliance Francophone des projets BOINC. Portail de l'alliance francophone des projets boinc). `http://www.boinc-af.org/content/view/49/128/`.

[32] R. Duan, R. Prodan, and T. Fahringer. Performance and cost optimization for multiple large-scale grid workflow applications. In *SC '07 : Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–12, New York, NY, USA, 2007. ACM.

[33] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek. Askalon : A grid application development and computing environment. In *GRID '05 : Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 122–131, Washington, DC, USA, 2005. IEEE Computer Society.

[34] C. Ferdinand and R. Wilhelm. On predicting data cache behavior for real-time systems. In *LCTES '98 : Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems*, pages 16–30, London, UK, 1998. Springer-Verlag.

[35] Open Grid Forum. Open grid forum homepage. http ://www.ogf.org/.

[36] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-g : A computation management agent for multi-institutional grids. In *Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*, August 2001.

[37] M. R. Garey and D. S. Johnson. *Computers and Intractability ; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[38] N. Garnier, A. Friedrich, R. Bolze, E. Bettler, L. Moulinier, C. Geourjon, J. D. Thompson, G. Deleage, and O. Poch. Magos : Multiple alignment and modelling server. *Bioinformatics*, 22(17) :2164–2165, 2006.

[39] A. Gerasoulis and T. Yang. A comparison of clustering heuristics for scheduling directed acyclic graphs onto multiprocessors. *Parallel and Distributed Computing*, 16(4) :276–291, 1992.

[40] T. Glatard. *Description, Deployment and Optimization of Medical Image Analysis Workflows on Production Grids*. PhD thesis, Université de Nice Sophia-Antipolis, Sophia-Antipolis, November 2007.

[41] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec. Flexible and Efficient Workflow Deployement of Data-Intensive Applications on Grids with MOTEUR. *International Journal of High Performance Computing and Applications (IJHPCA)*, 2008.

[42] C. A. Goble and D. C. De Roure. Myexperiment : Social networking for workflow-using e-scientists. In *WORKS '07 : Proceedings of the 2nd workshop on Workflows in support of large-scale science*, pages 1–2, New York, NY, USA, 2007. ACM.

[43] C. Gomez. *Engineering and Scientific Computing with Scilab*. Birkhauser Boston, 1998.

[44] R. L. Graham. Bounds on multiprocessing timing anomalies. *Siappmat*, 17 :263–269, 1969.

[45] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Ronnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. In *Ann. Discrete Math*, pages 287–326, 1979.

[46] R. W. Hockney. The communication challenge for mpp : Intel paragon and meiko cs-2. *Parallel Comput.*, 20(3) :389–398, 1994.

[47] IBM. Tivoli workload scheduler loadleveler. http ://www-03.ibm.com/servers/eserver/clusters/software/loa-dleveler.html.

[48] Internet. Workflow management coalition. `http://www.wfmc.org`.

[49] M. Iverson and F. Ozguner. Dynamic, competitive scheduling of multiple dags in a distributed heterogeneous environment. In *HCW '98 : Proceedings of the Seventh Heterogeneous Computing Workshop*, page 70, Washington, DC, USA, 1998. IEEE Computer Society.

[50] M. A. Iverson, O. Gregory, and J. Follen. Parallelizing existing applications in a distributed heterogeneous environment. In *4th Heterogeneous Computing Workshop (HCW '95)*, pages 93–100, 1995.

[51] R. Jain, D. Chiu, and W. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. *ArXiv Computer Science e-prints*, September 1998.

[52] M. Jan. *JUXMEM : un Service de partage transparent de données pour grilles de calcul fondé sur une approche pair-à-pair*. PhD thesis, Université de Rennes 1, 2006.

[53] E. Jeannot. *Algorithms and Protocols for Data and Computation Management in Distributed and Heterogeneous Environments*. HDR, Université Henri Poincaré, Nancy, 2007.

[54] A. H. G. Rinnooy Kan. *Machine Scheduling Problems : Classification, Complexity and Computation*. Martinus Nijhoff, The Hague, 1976.

[55] T. Kielmann, H. E. Bal, and K. Verstoep. Fast measurement of LogP parameters for message passing platforms. *Lecture Notes in Computer Science*, 1800 :1176–1183, 2000.

[56] S.J. Kim and J.C. Browne. A general approach to mapping of parallel computation upon multiprocessor architectures. In *Int'l Conf. Parallel Processing*, volume 2, pages 1–8, 1988.

[57] D. Kondo, G. Fedak, F. Cappello, A. A. Chien, and H. Casanova. Characterizing resource availability in enterprise desktop grids. *Future Gener. Comput. Syst.*, 23(7) :888–903, 2007.

[58] K. Krauter, R. Buyya, M, and Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Int. Journal of Software : Practice and Experience*, Vol 32, No. 2, Feb 2002.

[59] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4) :406–471, 1999.

[60] S. Lacour. *Contribution à l'Automatisation du Déploiement d'Applications sur des Grilles de Calcul*. PhD thesis, Université de Rennes 1, 2005.

[61] L. Lefèvre, J.-P. Gelas, and A.-C. Orgerie. How an experimental grid is used : the grid5000 case and its impact on energy usage. Poster CCGrid2008 : 8th IEEE International Symposium on Cluster Computing and the Grid, Lyon, France, May 2008.

[62] Arnaud Legrand, Alan Su, and Frédéric Vivien. Minimizing the stretch when scheduling flows of divisible requests. *Journal of Scheduling*, 2008. To appear.

[63] Wikipedia l'encyclopédie libre. grille informatique. `http://fr.wikipedia.org/wiki/Grille_de_calcul`.

[64] Wikipedia l'encyclopédie libre. intergiciel. `http://fr.wikipedia.org/wiki/Intergiciel`.

[65] J. E. Lennard-Jones. Cohesion. *Proceedings of the Physical Society*, 43 :461–482, September 1931.

[66] S. H. Low. A duality model of tcp and queue management algorithms. *IEEE/ACM Trans. Netw.*, 11(4) :525–536, 2003.

[67] S. Majithia, M. Shields, I. Taylor, and I. Wang. Triana : A graphical web service composition and execution toolkit. In *ICWS '04 : Proceedings of the IEEE International Conference on Web Services*, page 514, Washington, DC, USA, 2004. IEEE Computer Society.

[68] M. Mezmaz, N. Melab, and E-G. Talbi. A grid-enabled branch and bound algorithm for solving challenging combinatorial optimization problems. In *In Proc. of 21th IEEE Intl. Parallel and Distributed Processing Symp.*, Long Beach, California, March 2007.

[69] J. Mintseris, K. Wiehe, B. Pierce, R. Anderson, R. Chen, J. Janin, and Z. Weng. Protein-protein docking benchmark 2.0 : An update. *Proteins : Structure, Function, and Bioinformatics*, 60(2) :214–216, 2005.

[70] H. Nakada, M. Sato, and S. Sekiguchi. Design and implementations of ninf : towards a global computing infrastructure. *Future Generation Computing Systems, Metacomputing Issue*, 15(5-6) :649–658, 1999. `http://ninf.apgrid.org/papers/papers.shtml`.

[71] H. Nguyen, G. Berthommier, A. Friedrich, L. Poidevin, R. Ripp, L. Moulinier, and O. Poch. Introduction du nouveau centre de données biomédicales décrypthon. In *CORIA 2008, COnférence en Recherche d'Information et Applications*, March 2008.

[72] International Standards Organisation. Osi reference model. http ://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip.

[73] J.-F. Pineau, Y. Robert, and F. Vivien. The impact of heterogeneity on master-slave scheduling. *Parallel Comput.*, 34(3) :158–176, 2008.

[74] F. Plewniak, L. Bianchetti, A. Brelivet, Y.and Carles, F. Chalmel, O. Lecompte, T. Mochel, L. Moulinier, A. Muller, J. Muller, V. Prigent, R. Ripp, J.-C. Thierry, J. D. Thompson, N. Wicker, and O. Poch. Pipealign : a new toolkit for protein family analysis. *Nucl. Acids Res.*, 31(13) :3829–3832, 2003.

[75] C. Pouzat, M. Delescluse, P. Viot, and J. Diebolt. Improved spike-sorting by modeling firing statistics and burst-dependent spike amplitude attenuation : a Markov chain Monte Carlo approach. *J Neurophysiol*, 91(6) :2910–2928, 2004. Available from : `http://intl-jn.physiology.org/cgi/content/abstract/91/6/2910`.

[76] EGEE project. Enabling grids for e-science web page. http ://project.eu-egee.org.

[77] The R project Team. R-project : R a software environment for statistical computing and graphics. `http://www.r-project.org/`.

[78] M. Quinson. *Découverte automatique des caractéristiques et capacités d'une plate-forme de calcul distribué*. PhD thesis, École normale supérieure de Lyon, December 2003.

[79] C.R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, New York, NY., 1995.

[80] RENATER. Réseau national de télécommunications pour la technologie l'enseignement et la recherche. http ://www.renater.fr.

[81] S. Sacquin-Mora, A. Carbone, and R. Lavery. Identification of protein interaction partners and protein-protein interaction sites. *Journal of Molecular Biology*, 382(5) :1276 – 1289, 2008.

[82] V. Sarkar. *Partitioning and Scheduling Parallel Programs for Multiprocessors*. MIT Press, Cambridge, MA, USA, 1989.

[83] M. Sato, T. Boku, and D. Takahasi. Omnirpc : a grid rpc system for parallel programming in cluster and grid environment. In *Proceedings of CCGrid2003*, pages 206–213, Tokyo, May 2003.

[84] E. Seidel, G. Allen, A. Merzky, and J. Nabrzyski. Gridlab : a grid application toolkit and testbed. *Future Gener. Comput. Syst.*, 18(8) :1143–1153, 2002.

[85] K. Seymour, C. Lee, F. Desprez, H. Nakada, and Y. Tanaka. The end-user and middleware apis for gridrpc. In *Workshop on Grid Application Programming Interfaces, In conjunction with GGF12*, Brussels, Belgium, September 2004.

[86] Z. Shi and J. J. Dongarra. Scheduling workflow applications on processors with different capabilities. *Future Gener. Comput. Syst.*, 22(6) :665–675, 2006.

[87] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel Distrib. Syst.*, 4(2) :175–187, 1993.

[88] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147 :195–197, 1981.

[89] DataGrid Team. Europeen datagrid. `http://eu-datagrid.web.cern.ch/eu-datagrid/`.

[90] The BOINC Team. The berkeley open infrastructure for network computing. `http://boinc.berkeley.edu/`.

[91] The Condor Team. Condor : High throughput computing. `http://www.cs.wisc.edu/condor`.

[92] The Condor Team. Dagman : Directed acyclic graph manager. `http://www.cs.wisc.edu/condor/dagman`.

[93] The DIET Team. Diet : Distributed interactive engineering toolbox). `http://graal.ens-lyon.fr/DIET`.

[94] The DIET team. Diet user's manual. http ://graal.ens-lyon.fr/DIET/UsersManualDIET2.3/index.html.

[95] The DIET team. Godiet : a deployment tool for diet. http ://graal.ens-lyon.fr/ diet/godiet.html.

[96] The DIET team. Vizdiet : a vizual representation of diet. http ://graal.ens-lyon.fr/ diet/vizdiet.html.

[97] The Ganglia team. Ganglia : a scalable distributed monitoring system. http ://ganglia.wiki.sourceforge.net/.

[98] The Grid'5000 Team. Grid'5000 web page. http ://www.grid5000.org/.

[99] The Pegasus Team. Ensemble manager : Coordinate and efficiently handle multiple workflows. `http://pegasus.isi.edu/ensemble/`.

[100] The Taverna Team. Taverna workbench : a software tool for designing and executing workflows. `http://taverna.sourceforge.net/`.

[101] The WCG team. World community grid. http ://www.worldcommunitygrid.org/.

[102] the Beowulf team. Beowulf : Scalable performance clusters based on commodity hardware,. `http://www.beowulf.org/`.

[103] the Kadeploy team. Kadeploy : a fast and scalable deployment system. `http://kadeploy.imag.fr/`.

[104] the OAR team. Oar : Resource management system for high performance computing. `http://oar.imag.fr/index.html`.

[105] the OpenPBS team. Openpbs : Open source workload management software. `http://www.pbsgridworks.com`.

[106] the Platform team. Platformlsf : Managing and accelerating batch workload processing. `http://www.platform.com/Products/platform-lsf`.

[107] the SGE team. Sge : Sun grid engine. `http://gridengine.sunsource.net/`.

[108] the TORQUE team. Torque : Open source resource manager. http ://www.clusterresources.com/pages/products/torque-resource-manager.php.

[109] the WISDOM team. Wisdom : Wide in silico docking on malaria. `http://wisdom.eu-egee.fr/`.

[110] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Task scheduling algorithms for heterogeneous processors. In *HCW '99 : Proceedings of the Eighth Heterogeneous Computing Workshop*, page 3, Washington, DC, USA, 1999. IEEE Computer Society.

[111] H. Topcuouglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(3) :260–274, 2002.

[112] J. W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, New York, 1970.

[113] F. Wolf and R. Ernst. Execution cost interval refinement in static software analysis. *J. Syst. Archit.*, 47(3-4) :339–356, 2001.

[114] I. Ahmad Y. Kwok. Benchmarking the task graph scheduling algorithms. In *IPPS '98 : Proceedings of the 12th. International Parallel Processing Symposium on International Parallel Processing Symposium*, page 531, Washington, DC, USA, 1998. IEEE Computer Society.

[115] T. Yang and A. Gerasoulis. Dsc : Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, 5 :951–967, 1994.

[116] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.*, 34(3) :44–49, 2005.

[117] M. Zacharias. Protein-protein docking with a reduced protein model accounting for side-chain flexibility. *Protein Sci*, 12(6) :1271–1282, 2003.

[118] H. Zhao and R. Sakellariou. An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm. In *Euro-Par*, volume Lecture Notes in Computer Science Vol. 2790. Springer-Verlag, 2003.

[119] H. Zhao and R. Sakellariou. Scheduling multiple dags onto heterogeneous systems. In *Proceedings of the 15th Heterogeneous Computing Workshop (HCW)*, Rhodes Island, Greece, April 2006.