utbm
université de technologie
Belfort-Montbéliard

CDS
Centre de Données
astronomiques de Strasbourg

11, rue de l'Université
67000 Strasbourg

ST40 – Stage professionnel

Développement d'un environnement de distribution de tâches

Thomas BUCHER – GI03

Automne 2003

Suiveurs en entreprise : François BONNAREL et Pierre FERNIQUE
Suiveur UTBM : Jaafar GABER

Remerciements

Tout d'abord, je tiens à remercier Jean-Marie HAMEURY, le directeur de l'Observatoire, ainsi que Françoise GENOVA, la directrice du CDS, pour m'avoir permis de joindre leur personnel durant ces quelques mois.

Ensuite, je remercie François BONNAREL et Pierre FERNIQUE, mes tuteurs, pour toutes les informations qu'ils m'ont apportées, pour les conseils qu'ils m'ont donnés, pour leur suivi, leur patience et leur intérêt porté sur le travail que j'ai réalisé.

D'autre part, je suis reconnaissant envers Thomas KELLER et Jean-Yves HANGOUET, les administrateurs, pour le temps qu'ils ont consacré à m'aider lors de problèmes réseau et d'administration.

Je remercie aussi André SCHAAFF, membre de l'équipe ViZieR, pour s'être impliqué dans la correction des différentes documentations et pour sa participation aux tests de mon système, de même que Christophe PICHON et Anaïs OBERTO pour leur collaboration commune à l'interfaçage de mon système avec leurs applications.

Par ailleurs, je tire mon chapeau à Marc WENGER, pour avoir répondu posément à plusieurs de mes questions concernant différents langages et concepts informatiques.

Enfin, je remercie tout le reste de l'équipe pour son accueil chaleureux et sa bonne humeur générale, ainsi que Benoît pour avoir partagé avec moi pendant 6 mois ses repas du midi et ses pauses café.

Sommaire

Introduction

Au terme de deux semestres d'études en cycle d'ingénieur à l'UTBM (Université Technologique de Belfort-Montbéliard) s'inscrit un stage professionnel d'environ 6 mois. Cette étape nous permet, étudiants, de mieux apprécier le milieu professionnel en plein milieu de notre cursus.

Ayant déjà réalisé un stage dans un laboratoire universitaire, je n'ai pas hésité à choisir de nouveau le même environnement. En effet, les laboratoires universitaires ont l'avantage de donner de la liberté dans les choix de l'étudiant, tout en proposant un environnement à forte dominante intellectuelle : ce mélange permet l'épanouissement de l'étudiant ainsi qu'une formation accrue.

L'Observatoire Astronomique de Strasbourg, ainsi que le sujet de stage proposé m'ont permis de réaliser du début à la fin un projet intéressant, correspondant à mes ambitions dans le monde informatique.

Je commencerai tout d'abord par décrire le lieu d'accueil et le contexte du sujet, ensuite j'expliquerai le sujet ainsi que les objectifs plus en détails, puis je montrerai ce que j'ai réalisé et enfin je synthétiserai le travail effectué et conclurai.

1 Présentation du lieu d'accueil

1.1 L'Observatoire Astronomique de Strasbourg

L'Observatoire astronomique de Strasbourg est un Observatoire des sciences de l'Univers, et une unité mixte de recherche de l'Université Louis Pasteur et du CNRS.

En tout, 70 personnes y travaillent dans les secteurs de la recherche, de l'enseignement universitaire et de la diffusion des données astrophysiques auprès des astronomes professionnels du monde entier par l'intermédiaire du Centre de Données astronomiques de Strasbourg (CDS).

Le planétarium, composante de l'Observatoire, est chargé de la diffusion des connaissances auprès du grand public. Il propose :

- un spectacle disposant d'un environnement immersif plongeant le spectateur sous un ciel simulé,
- une crypte aux étoiles, lieu de découvertes et d'activités ludiques sur le thème général de l'espace et de l'astronomie, et
- la visite de la grande coupole et de sa lunette astronomique.

L'Observatoire a été inauguré en 1881 dans le cadre de la nouvelle université de Strasbourg en application de l'édit de l'empereur Guillaume I du 28 avril 1872. Sa grande coupole abrite une lunette équatoriale, dotée d'un objectif de 49cm de diamètre et de 7m de focale. Cet instrument, qui était à la pointe de la technologie à la fin du XIX^{e} siècle, n'est plus utilisé par des astronomes professionnels, mais reste un important objet patrimonial.

1.2 Les thèmes de recherche à l'Observatoire

Les équipes de recherche de l'Observatoire suivent quatre thèmes dominants :

- **L'Astrophysique des Hautes Énergies**
  Les astres effondrés, comme les trous noirs ou certaines étoiles très denses (étoiles à neutrons, naines blanches) sont le siège de phénomènes qui comptent parmi les plus violents de l'Univers. Les conditions physiques qui y règnent sont inatteignables sur Terre. L'équipe Hautes Énergies s'intéresse aux phénomènes de capture de la matière, et de son éjection sous forme de jets par ces astres compacts.

- **Le Traitement massif de données**
  Les chercheurs de l'Observatoire s'appuient sur les compétences des personnels du CDS et sur les données auxquelles le centre donne accès pour comprendre, à partir de très grands relevés du ciel, la structure de l'Univers qui nous entoure. Le CDS mène également des activités de recherche technologique sur de nouvelles méthodes d'analyse, d'organisation, de traitement et de diffusion de grandes masses de données, qui peuvent être réparties sur plusieurs sites différents.

- **Galaxies**
  Les chercheurs de l'Observatoire étudient l'évolution des galaxies en analysant les populations d'étoiles qui les constituent. La compréhension des mouvements qui animent les

étoiles de notre galaxie, la Voie Lactée, dus à l'attraction gravitationnelle, permet de retrouver la trace des petites galaxies qui ont formé notre galaxie en s'agglutinant. Elle permet aussi de mettre en évidence l'existence de « matière noire », qui représente l'essentiel de la masse de l'Univers.

- **Physique Stellaire**

  Les étoiles sont acteurs de l'évolution des galaxies auxquelles elles appartiennent. Les recherches de cette équipe s'appuient sur l'observation et la modélisation des rayonnements émis. Le but est de comprendre les processus physiques qui gouvernent la structure et l'évolution des étoiles, ainsi que leurs liens avec le milieu interstellaire. L'étude des étoiles doubles permet d'aborder le mécanisme de leur formation et se prolonge dans le domaine des planètes extra solaires.

1.3 Le Centre de Données astronomiques de Strasbourg

Ce service a été créé en 1972 sous le nom de Centre de Données Stellaires, et est géré actuellement par un personnel composé de 10 chercheurs, 8 ingénieurs, 5 techniciens et plusieurs collaborateurs à contrat temporaire.

Le rôle du CDS est d'archiver une masse considérable de données d'observations astronomiques, de les organiser, de les hiérarchiser et d'établir des liens avec des services distants. C'est une importante valeur ajoutée que le CDS apporte à ces données d'observations.

Ainsi, ce service est utilisé quotidiennement par les astronomes du monde entier pour leur travail de recherche. Par ailleurs, les grandes agences qui gèrent les moyens d'observation astronomique au sol ou dans l'espace (l'Observatoire européen austral, l'Agence spatiale européenne, le CNES, la NASA, etc.) s'en servent également pour préparer les futures missions et identifier les sources observées par les télescopes.

De plus, le CDS a signé des accords d'échanges internationaux avec plusieurs organismes :

- la NASA (Etats-Unis),
- le National Astronomical Observatory (Japon),
- l'Académie des Sciences de Russie,
- le réseau PPARC Starlink au Royaume-Uni,
- l'Observatoire de Beijing (Chine),
- l'Université de Porto Allegre (Bresil),
- l'Université de La Plata (Argentine) et
- l'InterUniversity Center for Astronomy and Astrophysics (Inde).

Le CDS joue et a joué un rôle dans d'importantes missions astronomiques spatiales, contribuant aux catalogues d'étoiles guides (EXOSTAT, IRAS, Hipparcos, HST, ISO, SAX), aidant à identifier les sources observées (Hipparcos, Tycho, ROSAT, SIGMA) ou en organisant l'accès aux archives. Le CDS contribue aussi au XMM Survey Science Center, sous la responsabilité de l'équipe « Hautes Énergies » de l'Observatoire de Strasbourg.

Les 3 services majeurs proposés par le CDS sont les suivants :

- **SIMBAD**

  C'est une base de données de référence pour l'identification et la bibliographie de plus de 3 millions d'objets astronomiques. La mise en place d'une version accessible sur le web permet de développer progressivement les liens avec d'autres services, journaux en lignes, archives d'observatoires, images, etc. SIMBAD est utilisée dans pratiquement tous les grands systèmes d'accès aux archives et dans le service bibliographique ADS (Astrophysics Data System), comme serveur d'identification. Il est ainsi possible de retrouver, à partir d'un nom d'objet, toute la bibliographie de celui-ci. La base reçoit près de 15000 requêtes par jour.

- **VIZIER**

  Ce service permet d'effectuer des recherches dans plus de 4000 catalogues et tables publiés, dont certains comportent plusieurs centaines de millions d'entrées. Ces catalogues contiennent tous les objets astronomiques répertoriés dans le ciel. Le nombre de requêtes est du même ordre que pour SIMBAD.

- **ALADIN**

  Aladin est l'atlas interactif du ciel. Il permet de visualiser n'importe quelle partie du ciel sous forme d'images astronomiques ; d'origines, de longueurs d'ondes et de résolutions diverses. Outre la simple visualisation d'images, ce service permet de superposer ces images de référence du ciel avec le contenu des bases de données du CDS ou d'autres centres. D'autre part, il propose d'autres fonctionnalités techniques permettant d'extraire de façons variées l'informations contenue dans les images (courbes de niveaux, détection d'objets, etc.). Le serveur accuse environ 2000 requêtes par jour.

Le CDS s'engage aujourd'hui dans le projet d'Observatoire virtuel international, dont il est un acteur majeur. L'objectif est de donner un nouvel essor à la recherche astronomique européenne en fédérant les observations de grands observatoires au sol et dans l'espace qui sont stockées sous forme numérique dans des banques de données diverses. L'Observatoire virtuel permettra de reconstruire le ciel à toutes les longueurs d'onde et construira une base de travail fantastique pour les chercheurs.

2 Présentation du sujet

2.1 Contexte

De manière simplifiée, Aladin possède une banque de plusieurs millions d'images astronomiques (gérée par le SGBD Objectivity – Système de Gestion de Base de Données objet), stockée sur une baie RAID 7 d'environ 4To branchée en Firewire.

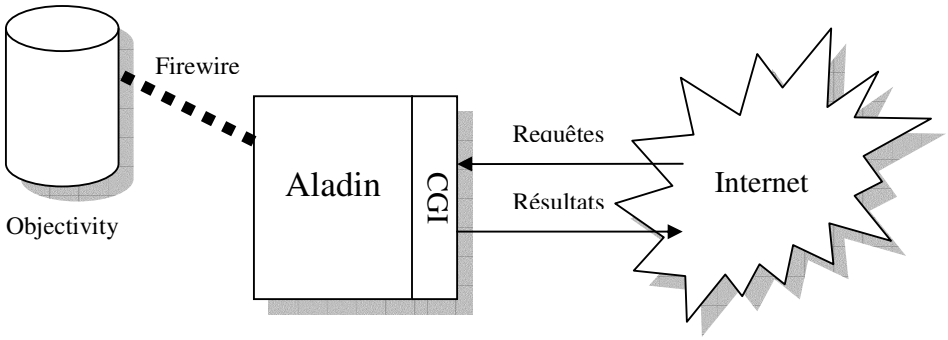

Figure 1. Structure du service Aladin

Le rôle principal du serveur est de reconstruire à la volée des images pour des régions du ciel, processus qui suit les étapes suivantes :

- analyse de la requête,
- récupération des images recouvrant la région demandée,
- décompression éventuelle de ces images,
- extraction des portions concernées,
- prétraitements éventuels (rééchantillonnage, ajustement de la dynamique, etc.)
- concaténation des différentes portions pour crée l'image finale,
- post-traitements éventuels (déconvolution, extraction d'objets, combinaison avec d'autres images),
- codage de l'image finale dans le format requis,
- compression de l'image finale si nécessaire.

La manière la plus courante d'interroger Aladin est d'utiliser son interface HTML (disponible sur http://aladin.u-strasbg.fr/AladinPreviewer) ou alors d'utiliser AladinJava, une application graphique écrite en Java, proposant de nombreuses fonctionnalités liées à tous les services du CDS.

Avec le nombre croissant de requêtes qu'Aladin doit traiter quotidiennement, la machine serveur se trouve souvent surchargée, et les utilisateurs sont alors confrontés à des temps d'attente qui peuvent être parfois relativement longs (selon les traitements demandés). L'équipe Aladin a donc décidé d'améliorer les performances du serveur, et trois solutions se présentaient alors :

- Trouver un moyen pour accélérer un code déjà optimisé : cette solution implique une refonte du serveur, chose non envisagée pour l'instant.
- Investir dans une machine plus puissante que l'actuelle (Sun quadri-processeur) : cette solution est très coûteuse, donc utilisée que si nécessaire.

- Mettre en place une ferme de machines (de simples PC), moins coûteuse qu'un seul gros serveur, qui déchargeront Aladin de certains traitements.

Ce projet d'évolution a été initié pas deux personnes qui seront alors mes tuteurs :
- Pierre FERNIQUE : ingénieur de recherche, responsable des outils graphiques associés au projet Aladin, et
- François BONNAREL : ingénieur de recherche, responsable du projet Aladin et particulièrement de la base d'images.

2.2 Sujet proposé

Après élimination de la solution d'optimisation et de l'achat d'un nouveau serveur coûteux, la solution de la ferme (cluster) de calcul a été retenue et a donné lieu au sujet de stage qui m'a été proposé :

> *« Mettre en œuvre un cluster de calcul pour le serveur d'images Aladin. Étude de faisabilité, évaluation des solutions logicielles, étude et programmation d'un gestionnaire de tâches adéquat (MPI), mise en place d'une plate-forme prototype et études de performances. »*

Par ailleurs, l'utilisation de fermes de calcul est très à la mode dans le monde scientifique, car peu chère et relativement performante. D'autre part, le concept de ferme de calcul s'intègre particulièrement dans le projet AVO (Astronomical Virtual Observatory) ainsi que dans GLOBUS, le projet de grille de calcul internationale.

J'aurai alors à ma disposition 3 nouveaux PCs, des Athlons XP 2400+ (en boîtier industriel 19''), dotés de 512Mo de RAM, 80Go de disque dur et d'une carte réseau 100Mbs. Ce ne sera qu'à titre d'essais, et d'autres machines pourraient être achetées par la suite si les résultats s'avèrent concluants.

Typiquement, une tâche pouvant être accélérée via un cluster est le mosaïquage. Le ciel est découpé en une multitude d'images ; ce sont ces images que contient la base de données d'Aladin. Lorsqu'un utilisateur demande une région du ciel autour d'un point précis, alors le serveur récupère les 4 images qui « entourent » ce point. Chaque image sera décompressée individuellement (4 traitements pouvant donc être faits en parallèle), puis elles seront « ressoudées » et l'image résultante sera de nouveau compressée. Dans ce cas, on tire parti du fait que 4 traitements peuvent être faits en parallèle, sur 4 machines différentes, alors que sur Aladin ils seront soit séquentiels, soit concurrents en terme de ressources.

2.3 Existant et expérience de l'équipe

L'idée d'utiliser un cluster de calcul n'est pas nouvelle au CDS. L'équipe de VizieR a déjà installé une grappe de 6 PC sous CLIC, une distribution Linux spécialisée dans le « clustering ». Cette ferme aurait du servir à décharger le serveur VizieR, et à augmenter sa capacité de stockage en répartissant de gros catalogues astronomiques sur chaque machine. Ainsi, chaque machine est spécialisée dans la recherche sur un catalogue précis. Les traitements auraient dû être faits en parallèle, grâce à une technologie nommée MPI (Message Passing Interface) qui sera expliquée par la suite.

Ce système a été mis au point par un stagiaire, qui a fait une étude de faisabilité et développé un prototype. Malheureusement, il s'est avéré que les performances du prototype n'atteignaient pas celles qui étaient attendues.

2.4 Déroulement du stage

Déterminer un planning n'est pas choses aisée, lorsque que l'on ne connaît pas encore les technologies que nous serons amenés à utiliser. Dans certaines entreprises, un cahier des charges strict est donné au stagiaire, qui devra alors le suivre à la lettre. Cela lui permet de ne pas « s'égarer », mais lui enlève aussi une certaine autonomie. Dans mon cas, on m'a simplement donné le sujet, avec les 3 étapes majeures à réaliser : étude et analyse, prototypage, développement final.

2.4.1 Planning prévisionnel

Des 3 étapes principales de mon stage, j'ai pu déduire un planning prévisionnel grossier, découpé en semaines sur un total de 24 semaines.

Durée	Réalisations
1ère semaine	➢ familiarisation avec l'environnement ➢ familiarisation avec les technologies existantes
2ème à 5ème semaine	➢ analyse des différentes possibilités ➢ étude des technologies possibles ➢ essais avec ces technologies
6ème à 10ème semaine	➢ développement d'un prototype ➢ essais avec ce prototype ➢ décision de rester sur ce choix
11ème à 20ème semaine	➢ développement d'une version finale ➢ essais et tests de charge ➢ correction des bugs
21ème à 24ème semaine	➢ rédaction des documentations ➢ corrections et mises au point de la version finale ➢ rédaction du rapport de stage

2.4.2 Planning réel

Il va de soi que le planning réel ne correspond pas exactement au planning prévisionnel, étant donné que je n'avais pas de contrainte sur mon organisation ; l'essentiel dans un laboratoire de recherche, c'est que le travail soit fait. Ainsi, mon stage c'est déroulé plus ou moins de la manière suivante :

Durée	Réalisations
1ère semaine	➢ familiarisation avec l'environnement ➢ familiarisation avec les technologies existantes
2ème à 3ème semaine	➢ étude de la distribution CLIC ➢ étude des logiciels de distribution de tâches (OpenPBS, LFS, Queue) + essais avec OpenPBS ➢ analyse de technologies dédiées au clustering (MPI, PVM)
4ème semaine	➢ réalisation de quelques essais avec MPI ➢ réalisation de quelques essais avec OpenPBS
5ème semaine	➢ installation matérielle du cluster prototype sous Linux ➢ décision de développer notre propre environnement de distribution de tâches
6ème à 10ème semaine	➢ développement du prototype ➢ réalisation de quelques essais concluants
11ème à 18ème semaine	➢ application du prototype à un cas réel : Marsiaa ➢ rajout de fonctionnalités au prototype et fiabilisation → vers une version exploitable ! ➢ finalisation et fiabilisation d'une grande partie du système
19ème à 20ème semaine	➢ interfaçage de l'environnement avec un service d'Aladin ➢ interfaçage de l'environnement avec des requêtes de VizieR ➢ migration vers un mode d'installation GNU → facilité d'installation sur toute plateforme Linux/Unix
21ème à 24ème semaine	➢ rédaction des documentations (Utilisateur, Administrateur, CDS) ➢ corrections et mises au point de la version finale ➢ ajout de la fonctionnalité d'items composés ➢ passage à un mode « toujours connecté » des exécuteurs et du serveur ➢ rédaction du rapport de stage

3 Recherche de technologies dédiées au clustering

Avant de se lancer dans la réalisation d'un système permettant de distribuer des tâches sur un réseau, il est utile de se renseigner sur l'état de l'art dans le domaine. Pour cela, il faut prendre en compte 3 composantes : les plateformes dédiées au calcul distribué, les langages et spécifications dédiées à la distribution, et enfin les logiciels qui, concrètement, servent à faire de la distribution de tâches. Si un logiciel correspondait exactement aux besoins du CDS, alors je n'aurais pas à le réaliser, mais simplement l'installer et le rendre exploitable de manière optimale. Nous verrons par la suite que ce ne sera pas le cas.

3.1 Qu'est-ce que le clustering ?

Strictement parlant, le clustering est la division d'une tâche complexe en plusieurs petites tâches tournant parallèlement sur une grappe de machines (appelée aussi ferme ou cluster). Par abus de langage, on utilise souvent ce mot aussi pour parler de distribution de tâches sur plusieurs machines.

Le calcul en parallèle permet d'exécuter une tâche, découpée en morceaux, plus rapidement, car plusieurs ordinateurs contribuent à la réalisation d'une même tâche. On cherche alors à accélérer l'exécution d'une seule tâche, tel un très gros calcul scientifique par exemple.

A contrario, il y a la distribution de tâche qui consiste à envoyer plusieurs tâches sur plusieurs machines différentes. Ceci n'accéléra pas le traitement d'une seule tâche, mais soulagera le serveur dans le cas de plusieurs tâches simultanées.

Pour lancer une tâche en plusieurs sous-tâches s'exécutant en parallèle, il faut qu'elle soit prévue pour, c'est-à-dire écrite de manière telle que plusieurs processus contribuent à sa réalisation en même temps. Dans le cadre de mon stage, le serveur Aladin n'est pas prévu pour être parallélisé, donc on ne fera que de la distribution. Cependant, la possibilité de lancer une tâche parallélisée via le système doit rester envisageable.

Il existe en tout 4 types de clusters, selon la ou les ressources partagées par les applications. Chaque cluster possède un rôle différent, dont est plus appropriée à un certain type de situation :

- Les clusters « de calcul scientifique » (HPC, High Performance Computing). Il s'agit d'un cluster où les nœuds cumulent leur puissance de calcul pour rivaliser avec les performances d'un supercalculateur. Le but d'un cluster de calcul est de faire coopérer les ordinateurs entre eux au sein d'un réseau en partageant les processeurs et la mémoire. On améliore ainsi le temps de calcul par rapport à des programmes qui exploiteraient localement le processeur et la mémoire d'une seule et même machine.

- Les clusters « haute disponibilité » (HA, High Avaibility). Ici on cherche à obtenir de la fiabilité par une redondance de machines, chacune pouvant prendre le relais de l'autre en cas de panne. Dans ce cas, le fonctionnement du cluster et l'assurance contre les pertes de données sont garantis au maximum.

- Les clusters de stockage : on cherche à grouper les espaces disques de plusieurs machines pour stocker d'énormes fichiers ne pouvant pas tenir sur un disque (ou plusieurs) d'une même machine.

- Les clusters de répartition de charge : ils sont fréquemment utilisés par les serveurs Web par exemple, en faisant traiter une requête d'un internaute par la machine la moins chargée à l'instant donné. La répartitions de charge est gérée par un algorithme de « load-balancing » plus ou moins complexe qui peut prévoir la charge des machines selon un certain nombre de paramètres.

Dans le cadre du stage, nous aurons à faire à la fois à un cluster de calcul (permettant de faire tourner certaines applications écrites en MPI), à un cluster de stockage (pour les gros catalogues de VizieR) et surtout à un cluster de répartition de charge (pour soulager le serveur Aladin).

3.2 Plateformes dédiées au clustering

Outre les supercalculateurs, peu de plateformes sont vraiment dédiées au calcul parallèle. En effet, pour effectuer du calcul parallèle, il faut une machine possédant plusieurs processeurs. Actuellement, les PCs sont monoprocesseurs, parfois biprocesseurs. C'est pourquoi, la mise en réseau de plusieurs PCs, donc de plusieurs processeurs, permet de « simuler » une machine multiprocesseur, avec l'inconvénient des temps de transferts de données sur le réseau.

CLIC (distribution Linux basée sur une Mandrake 9.1) propose de nombreux outils facilitant la mise en place d'une grappe de PCs, pouvant ainsi se comporter comme un gros calculateur. Comme l'équipe VizieR connaissait déjà cette technologie, on m'a tout de suite aiguillé dessus. J'ai donc étudié les possibilités de CLIC, et participé à une réinstallation complète de la ferme de VizieR.

L'avantage mis en avant par CLIC réside en sa facilité et sa rapidité à être déployé sur un grand nombre de machines : il suffit d'installer et configurer un nœud maître (le *Golden Node*), et tous les nœuds esclaves sont dupliqués à partir de celui-ci. Cette installation automatisée est mise en œuvre grâce à des cartes réseau compatibles avec le protocole PXE (Preboot Execution Environnement), et ne fonctionne qu'avec ce protocole.

3.3 API particulières au calcul parallèle

L'écriture d'un programme parallélisé n'est pas chose aisée car la communication interprocessus est parfois compliquée, et souvent dépendante de la plateforme. De plus, si on fait migrer une application parallèle prévue pour une machine vers un cluster, il faut réécrire la communication interprocessus pour pouvoir communiquer via le réseau.

La complexité de cette communication a poussé les développeurs à réaliser des API (Application Programming Interface) permettant de rendre transparente la communication interprocessus, que les processus soient sur une machine ou dispersés sur un réseau. De plus, ces API sont portables, donc permettent d'écrire simplement des applications portables sans se soucier de la plateforme ciblée.

3.3.1 LAM/MPI

LAM / MPI est l'implémentation de MPI proposée dans la distribution CLIC. MPI (Message Passing Interface) est comme son nom l'indique une interface de passage de message. MPI a été créée alors que la plupart des constructeurs réalisaient des API dépendantes de leurs systèmes. MPI avait alors pour but de permettre la réalisation d'applications multi processus portables sur toutes ces machines. Ainsi, le programmeur n'a plus à se soucier de la plateforme cible pour la création de son application, MPI assure qu'elle sera portable. Les bibliothèques proposent un large éventail de fonctionnalités, dont suivent les principales :

- un grand nombre de routines de communication point à point (bibliothèque la plus riche à ce jour),
- un grand nombre de routines de communication collective, pour la communication entre groupes de processus,
- un contexte de communication qui permet de développer des bibliothèques de calcul parallèle fiables,
- la possibilité de spécifier différentes topologies de communication,
- la possibilité de créer des types de données dérivés décrivant des messages contenant des données non contiguës.

Les programmes écrits avec MPI sont donc portables (peuvent être compilés sur toute plateforme possédant une implémentation de MPI), mais ne sont pas forcément interopérables (par exemple, un programme écrit avec LAM/MPI et tournant sous Linux ne peut pas forcément communiquer avec un programme écrit avec MPICH – une autre implémentation d'MPI - tournant sous Solaris). Actuellement, MPI est disponible pour les langages C, C++ et Fortran.

Concrètement, on lance une application MPI via la commande `mpirun`.

```
% mpirun –np 6 app
```

Cette commande va créer un environnement MPI et y lancera (dans cet exemple) 6 instances de l'application `app`, identifiées par les numéros 1 à 6. Ces processus pourront alors dialoguer très simplement, via les routines de communications MPI. Comme le montre la figure ci-dessous, les processus peuvent être lancés sur plusieurs machines différentes à travers le réseau, mais ceci n'entravera d'aucune manière leur communication.

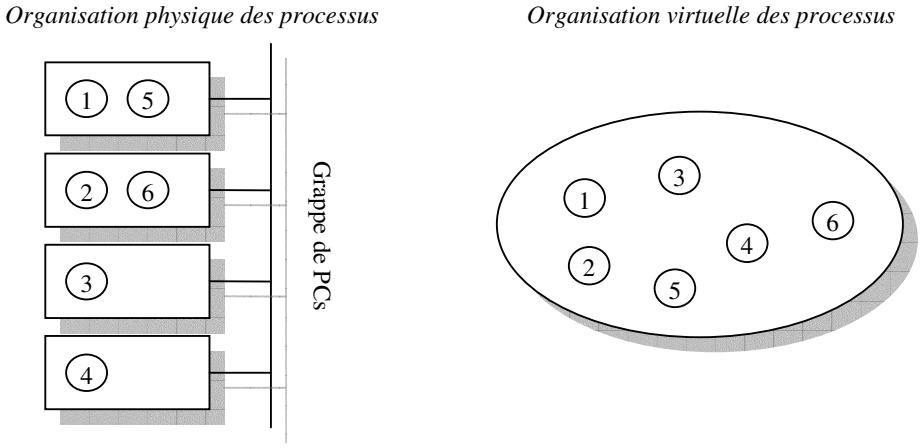

Figure 2. Création d'un environnement MPI

3.3.2 PVM

PVM (Parallel Virtual Machine) n'est pas seulement une API ; c'est un kit intégré d'outils et de bibliothèques qui émulent une structure de calcul flexible et hétérogène, basée sur des ordinateurs interconnectés d'architectures variées. L'objectif principal de PVM est de permettre d'utiliser un grand nombre d'ordinateurs différents qui travailleront en coopération pour faire du calcul parallèle ou concurrent. PVM dispose entre autre des fonctionnalités suivantes :

- Un pool d'hôtes configurable : les applications sont exécutées sur un ensemble de machines sélectionnées par l'utilisateur au lancement du programme. Les machines monoprocesseur et multiprocesseur (dont les architectures à mémoire partagée et à mémoire distribuée) peuvent faire partie de ce pool au même moment. Ce pool peut être modifié à tout instant, même lors de l'exécution de la tâche.

- Un accès transparent au matériel de la machine virtuelle: par exemple, un processus de la machine A peut utiliser le lecteur CD de la machine B comme si il était local.

- Une architecture générale de la machine ressemblant à un système Unix, c'est-à-dire que l'architecture est multi processus et multi utilisateurs.

- Un modèle de passage de message explicite : chaque processus communique et utilise les routines fournies par PVM.

- L'hétérogénéité : PVM supporte l'hétérogénéité en terme de réseaux, de machines et d'applications. PVM permet aussi d'échanger des données de types dépendant de la machine de manière portable, ce qui confère aux application une forte interopérabilité.

- La gestion des machines multiprocesseurs : PVM utilise les routines natives des architectures multiprocesseurs pour garder leur avantage en terme de performance.

La machine virtuelle est créée par un démon installé sur chaque machine, qui gère les droits utilisateurs ainsi que les applications lancées. Les langages supportés par PVM sont le C, le C++ et le Fortran.

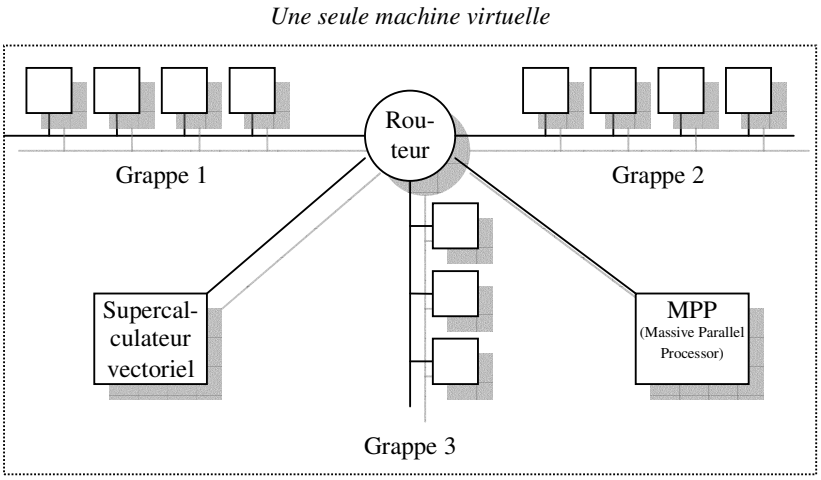

Figure 3. Architectures hétérogènes

3.4 Logiciels de distribution de tâches

Outre l'utilisation d'API permettant le calcul parallèle réparti de manière transparente sur le réseau, il est aussi possible de répartir des tâches sur un réseau grâce à un simple gestionnaire. Le l'utilisation général de ce genre d'utilitaires est de leur soumettre une tâche à effectuer, et celle-ci sera exécutée sur une machine adéquate. Trois logiciels correspondants aux besoins d'Aladin ont été étudiés, OpenPBS, LSF et Queue, ainsi qu'un environnement de répartition de charge automatique : OpenMosix.

3.4.1 OpenPBS

OpenPBS est la version originale du Portable Batch System. C'est un système de queues flexible, développé par la NASA dans le début des années 1990, permettant de faire du traitement différé (batch). Il fonctionne dans un environnement réseau multi plateformes (Unix/Linux). Voici quelques-unes des fonctionnalités de ce système :

- Une interface utilisateur graphique, permettant de soumettre des tâches et de suivre leur déroulement. Une interface en lignes de commande est aussi disponible.
- La gestion des priorités entre les différentes tâches.
- La gestion des dépendances entre les tâches : ordre d'exécution, synchronisations et exécutions conditionnelles sont possibles.
- Le transfert automatique des fichiers spécifiés par l'utilisateur.
- La gestion d'une ou plusieurs queues d'exécution.
- La possibilité d'utiliser différents algorithmes d'ordonnancement, tels que le *first-in, first-out* (par défaut), ou bien le *first-in, last-out*, etc.

3.4.2 Platform LSF

Platform LSF est un logiciel permettant la gestion et l'accélération des traitements de tâches, pour les applications gourmandes en calcul et/ou en données. De la même manière qu'OpenPBS, Platform LSF permet d'ordonnancer les tâches et de garantir leur complétion, le tout fonctionnant dans un environnement IT virtuel. Platform LSF est basé sur l'architecture VEM (Virtual Execution Machine), conçue pour fonctionner avec le projet Globus (mise en commun et partage de ressources informatiques à l'échelle internationale).

3.4.3 Queue

Queue est un système de balance de charge qui permet aux utilisateurs de contrôler leurs tâches distantes de manière intuitive et transparente. Ce contrôle est fait à l'aide d'un shell local. Queue peut être utilisé comme un remplacement local à *rsh* vers les machines d'un cluster homogène administré de manière unifiée. Un démon permet à l'utilisateur de contrôler la tâche qu'il a lancée comme si elle s'exécutait sur sa machine. De plus, Queue supporte aussi les traditionnels balances de charge et distribution de traitements, paramétrés selon un certain nombre de critères tels que la limite de temps d'exécution, la limite de stockage, un seuil de charge CPU, etc.

3.4.4 OpenMosix

OpenMosix est une extension de noyaux Linux, qui transforme un réseau de simples ordinateurs en un superordinateur pour applications Linux. Une fois qu'openMosix est installé, les nœuds du cluster communiquent entre eux et le cluster s'adapte de lui-même à la charge de travail. Ainsi, lorsqu'un processus tourne sur un nœud surchargé, il peut être délocalisé automatiquement vers un autre nœud plus adapté. OpenMosix effectue cette optimisation d'allocation de ressources continuellement, sans intervention quelconque de la part de l'utilisateur.

Ainsi, openMosix crée une plateforme de clustering rentable, rapide et fiable, qui peut être agrandie et adaptée à souhait. Avec openMosix' Auto Discovery, de nouveaux nœuds peuvent être ajouté alors que le cluster fonctionne déjà, et ils seront automatiquement reconnus et utilisés. Le cluster se comporte tel un SMP (Symmetric Multi-Processor), mais cette solution peut tout à fait être appliquée à un millier de nœuds qui eux-mêmes peuvent être des SMPs.

De plus, les applications destinées à fonctionner sous openMosix n'ont pas à être spécialement programmées pour, contrairement à celles de PVM par exemple. Comme toutes les extensions openMosix sont intégrées au noyaux, toute application Linux bénéficie automatiquement et de manière transparente du concept de calcul distribué d'openMosix. En effet, l'utilisateur n'a pas à « soumettre » une tâche au cluster ; le simple fait de lancer une application fera qu'elle sera prise en charge automatiquement par le cluster.

4 Analyse comparative des technologies

Maintenant que nous avons décrit les différentes technologies disponibles, passons à leur critique, en montrant leurs avantages et leurs inconvénients dans le cadre du sujet proposé. L'analyse de chaque technologie aboutit à un choix quant à son utilisation ; c'est le moment décisif à partir duquel on va se lancer dans l'élaboration d'un projet prototype, c'est pourquoi ces choix doivent être faits de manière objective après une analyse approfondie.

4.1 Choix de la plateforme utilisée

CLIC semblait être une distribution prometteuse pour l'élaboration d'un cluster de manière aisée. Malheureusement, une expérience négative nous en a prouvé le contraire. En effet, le cluster de l'équipe VizieR avait subit un dysfonctionnement qui faisait que le cluster complet devait être réinstallé. J'ai participé à cette réinstallation durant la troisième semaine de mon stage, et pendant deux jours l'équipe a essayé, en vain, d'utiliser le mode automatique d'installation. A priori, celui-ci n'a pas fonctionné suite à des problèmes DHCP. Ainsi, CLIC perd tout son avantage de simplicité, et devient une simple distribution proposant des packages dédiés au clustering. De plus, CLIC connaît aussi des problèmes de drivers qui doivent être outrepassés de manière manuelle et complexe. Enfin, pour que l'installation automatique fonctionne, les machines cibles doivent être matériellement identiques (disques de même capacité, mémoire de même taille, etc.) : c'est un contrainte très forte dans le cas d'un cluster, car, dans ce cas, il doit être strictement homogène.

Cette expérience infructueuse m'a montré qu'il ne valait mieux pas se concentrer sur une plateforme en particulier. Sachant que le réseau du CDS est relativement hétérogène (Unix, Linux et Windows), j'ai décidé de me diriger vers une solution portable (au moins Unix et Linux) afin de pouvoir utiliser le maximum de machines mises à disposition.

De plus, outre de désir d'utilisation optimale des ressources du CDS, je suis contraint de prévoir le serveur Aladin comme machine faisant partie du cluster. En effet, le code du serveur Aladin n'est actuellement prévu que pour Unix (Solaris) ; son portage vers un Linux n'est actuellement pas à l'ordre du jour, et ce serait beaucoup trop long pour que je puisse l'utiliser à temps pendant mon stage. C'est pourquoi j'ai été obligé de tenir compte d'au moins une machine Unix dans la structure du cluster à mettre en oeuvre.

4.2 Choix d'un logiciel existant

Tout d'abord, on peut exclure *Platform LSF* de la liste des logiciels envisageables, puisque celui-ci est payant. Le CDS souhaite plutôt accéder à des solutions gratuites, afin de tester les possibilités d'un cluster dans notre contexte. Après, si les performances du prototype sont convaincantes, il serait toujours possible d'investir dans un système encore plus performant.

Ensuite, étudions le cas de *Queue*. Ce système se prête bien au contexte, mais possède néanmoins un désavantage : il ne fonctionne que sur un cluster homogène. En effet, il n'est pas possible de mélanger à la fois des machines sous Unix et des machines sous Linux dans le même cluster. Cette limitation va à l'encontre de la contrainte d'hétérogénéité imposée par le code non portable d'Aladin.

OpenMosix semble très intéressant, de par la transparence qu'il propose. De manière simple, il suffirait d'installer openMosix sur toutes les machines du cluster, et de faire tourner simplement les tâches depuis n'importe quel nœud, et celles-ci seraient automatiquement délocalisées afin de préserver un équilibre des charges CPU.

Cependant, la régulation de charge réalisée par openMosix n'est effective que pour les processus actifs depuis un certain temps (une minute par exemple) ; or les tâches lancées par Aladin seront plus courtes pour la plupart, donc ne seront jamais déportées de la machine Aladin vers d'autres machines moins chargées.

De plus, étant donné qu'openMosix est une extension du noyau Linux, il n'est pas possible de l'utiliser avec le code d'Aladin.

On se rend alors compte que l'hétérogénéité (donc la portabilité des applications) est une contrainte forte, qui risque de poser de nombreux problèmes. On arrive alors à la dernière solution logicielle : *OpenPBS*. Celle-ci est entièrement portable donc colle parfaitement à notre besoin d'hétérogénéité. J'ai donc décidé d'installer OpenPBS sur les machines du cluster afin de tester ses fonctionnalités et ses performances. Les fonctionnalités correspondent bien aux besoins d'Aladin : on peut soumettre des tâches à un démon serveur (*pbs_server*), qui se charge de la faire effectuer à l'une des machines exécutrices (chacune contrôlée par un démon *pbs_mom*).

L'ensemble des ressources est monitoré par un démon nommé *pbs_mon*. De plus, au niveau du serveur, il est possible de mettre en place de queues d'exécution permettant d'ordonnancer les tâches, selon leur priorité ou leur destination par exemple. Plusieurs instances de serveurs peuvent exister, et le passage d'une queue à une autre peut se faire d'un serveur à un autre. Ainsi, on peut mettre en place une architecture complexe, déployée par exemple sur plusieurs centaines de machines... Mais dans notre cas, étant donné le nombre limité de machines du cluster, une simple queue en FIFO (first-in, first-out) et un seul serveur suffiraient.

Les fonctionnalités d'OpenPBS sont intéressantes, mais les performances ne sont pas au rendez-vous. En effet, les requêtes provenant d'Aladin sont des services rendus à des utilisateurs en mode interactif, c'est pourquoi le temps de réponse est très important. Le problème est qu'OpenPBS est un gestionnaire de batch (traitement par lot ou traitement différé), et n'assure en aucun cas le traitement instantané des tâches demandées. Ainsi, une tâche devant être exécutée en moins de 10 secondes par Aladin, sera exécutée en 1 minute par OpenPBS. En effet, ce logiciel est prévu pour des longues tâches, et ne se soucie pas du temps minime qu'il prend pour les mettre en place. Apparemment, la gestion des queues ainsi que les mises à jour des charges CPU ne sont pas fréquentes, mais cadencées par une horloge à période assez longue. Ces temps de latence ne sont pas tolérables pour Aladin, donc la solution OpenPBS est mise à l'écart.

Étant donné qu'aucune des solutions logicielles étudiées n'est adaptée au cas Aladin, il ne me reste plus qu'à développer moi-même un environnement de distribution de tâches, destiné à un réseau hétérogène.

4.3 Utilisation d'une API pour la réalisation

LAM / MPI est l'API fournie par défaut dans CLIC, et dès mon arrivée on m'a fait comprendre que je serai très certainement confronté à MPI pour mes développements. Après quelques jours de lecture et de tests, il s'est avéré que MPI est une API dédiée surtout au calcul parallèle, et non à la distribution de tâches. En effet, en environnement MPI est très stati-

que ; il est relativement difficile d'y insérer un nouveau processus dynamiquement. MPI n'est pas du tout adaptée pour la distribution de tâches comme on souhaite le faire avec Aladin. En terme de cluster, MPI est tout à fait adaptée pour un cluster de calcul, et non pas pour un cluster de répartition de charge, d'autant plus que MPI ne propose pas de système intégré de « load-balancing ».

En fait, j'ai concrètement exclu MPI après avoir fait des tests non concluants. En effet, j'avais alors déjà élaboré la structure du futur gestionnaire de tâches, et un algorithme en particulier ne pouvait être conçu en MPI ; étant donné que l'organisation des processus est statique en MPI, il n'est pas possible de dupliquer un processus (un serveur par exemple), et d'insérer ce processus fils dans l'environnement MPI. Au contraire, ce duplicata partage avec son père tous les moyens de communication et génère ainsi des perturbations néfastes.

Ainsi, MPI ne sera pas utilisée pour programmer le gestionnaire de tâches, mais pourra néanmoins servir à écrire des applications de calcul qui seront lancées par le gestionnaire.

PVM permet d'implémenter l'algorithme qui n'était pas implémentable avec MPI, mais possède un inconvénient majeur : des performances moindres. En effet, de par la gestion d'une multitude d'architectures pour pouvoir transformer une topologie matériellement hétérogène en une et une seule machine virtuelle, PVM connaît une perte de performance non négligeable lors des communications interprocessus. Or, la moindre seconde de gagnée est importante pour rendre les services d'Aladin plus rapides.

Dans les deux cas (MPI et PVM), l'API est destinée à faciliter majoritairement le développement d'applications de calcul parallèle, surtout celles qui communiquent beaucoup. La création d'un gestionnaire de tâche, comme nous le verrons, n'a rien à voir avec le calcul parallèle. En effet, nous verrons que la structure du gestionnaire tient en 3 démons, qui n'effectuent jamais de calcul, mais s'occupent juste de gérer soit un client, soit une tâche.

De plus, l'indépendance vis-à-vis d'une API permet de distribuer une application sur toute machine sans installation supplémentaire ; un plus lorsque qu'on a à faire avec un cluster hétérogène, ne disposant pas forcément de toutes les bibliothèques dont nous avons besoin.

5 Développement

Une fois la phase d'analyse des technologies disponibles finie, il est possible de commencer à concevoir le prototype de gestionnaire de tâches. Comme conclu, tout le projet sera recréé à partir de zéro. Ceci n'est pas tellement dérangeant dans le sens où le laboratoire préfère maîtriser totalement les outils qu'il utilise. De plus, fabriquer un outil du début jusqu'à la fin permet de coller au mieux aux besoins du moment, alors qu'un outil « extérieur » fera certaines choses, mais pas forcément tout ce que l'on voudrait. Enfin, maîtriser un outil, connaître son code, permet aussi de pouvoir le maintenir de manière aisée ; la maintenance de logiciels extérieurs n'est pas toujours simple, surtout si ceux-ci ne possèdent pas ou plus de support technique.

5.1 Structure de l'environnement de distribution

Pour résumer, nous avons besoin d'un gestionnaire de tâches, qui reçoit des requêtes provenant du client (Aladin), qui exécute les tâches demandées (en les distribuant sur le cluster) et qui renvoie enfin les résultats au client. Le schéma global est représenté par la figure cidessous. Aladin reçoit une requête (1), exécute ce qu'il peut et demande au gestionnaire d'exécuter certaines tâches (2). Le gestionnaire répartit alors ces tâches vers les machines du cluster (3) qui vont les exécuter, puis renvoyer les résultats au gestionnaire (4). Enfin, ce dernier renvoie les résultats à Aladin (5), qui les exploite pour pouvoir finir de répondre au client (6). Le gestionnaire sera un programme installé soit sur la machine Aladin, soit sur une autre machine afin de bien séparer le rôle de chaque élément du réseau.

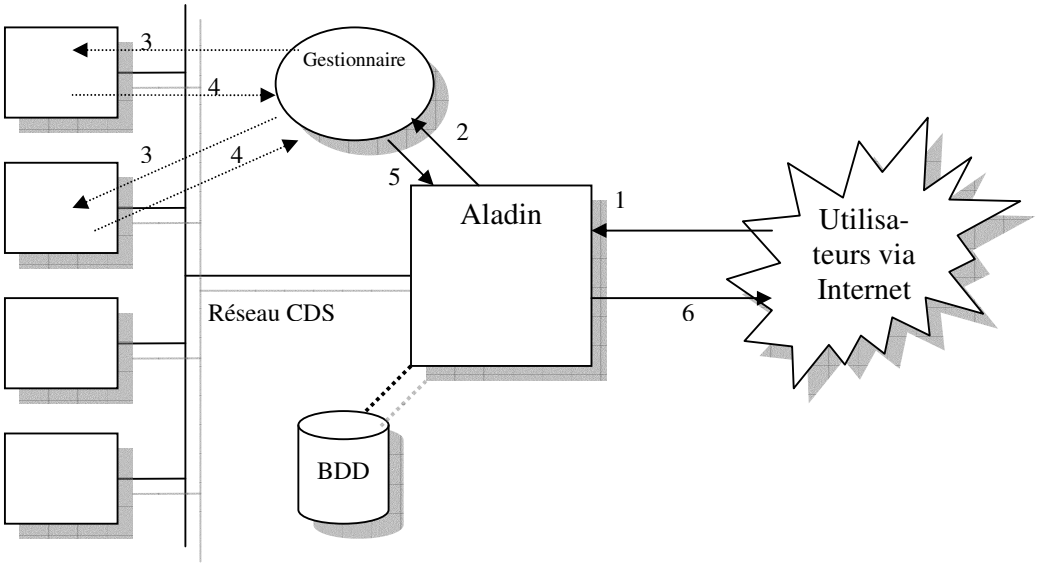

Figure 4. Schéma global de l'interaction Aladin - Gestionnaire

Sachant que le cluster sera hétérogène, certaines machines ne seront sûrement pas capables d'effectuer certaines tâches. Ainsi, il est intéressant de mettre en place un système de « disponibilité d'items ». Le gestionnaire ne distribuera les tâches (nécessitant certains items) qu'en fonction des items proposés par les machines exécutrices.

De plus, il faut mettre en place un système de répartition de charge, afin de ne pas distribuer les tâches toujours aux mêmes nœuds du cluster. Nous avons donc besoin d'un moniteur, qui va gérer une table contenant les informations de chaque machine du cluster ; ces informations pourront être la charge CPU actuelle, les items proposés, le nombre de tâches exécutées, etc.

Enfin, limiter le système à un seul gestionnaire n'est pas très pertinent. En effet, il serait intéressant que plusieurs gestionnaires puissent fonctionner pour le même cluster, en tirant parti d'un seul moniteur. Ainsi, il pourrait y avoir un gestionnaire spécialisé dans les requêtes provenant d'Aladin, et un autre pour celles provenant de VizieR (évolution que nous verrons plus tard). Le client devient alors n'importe qui, pas seulement le serveur Aladin, et les requêtes seront envoyées à partir d'une interface de soumission (le *submitter*). On obtient une architecture comme illustrée ci-dessous (les différents serveurs possibles et la pluralité de clients ne sont pas représentés):

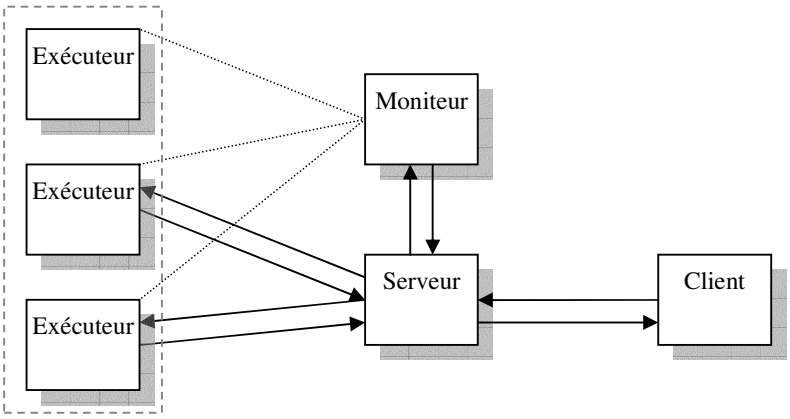

Figure 5. Structure générale de l'environnement de distribution

Lorsque le client soumet une tâche au serveur, celui-ci demande au moniteur l'adresse d'un exécuteur (ou plusieurs, selon le nombre de tâches). Le serveur ordonne alors à cet exécuteur d'effectuer la tâche donnée, attend les résultats et les renvoie au client. On voit que le moniteur garde une connexion active avec tous les exécuteurs, afin de garder une table de leurs informations, mise à jour fréquemment.

5.2 Choix du langage

Il existe trois langages possibles pour la réalisation d'un tel projet : le C, le C++ et le Java. D'autres langages, tels que le Fortran ou le Perl ne sont pas envisageables, de part leur faible performance ou leurs routines inadaptées.

Le Java est un langage très pratique pour développer rapidement des logiciels fiables. Cependant, rares sont les serveurs écrits en Java, car ce langage n'est pas aussi performant que C ou C++. De plus, ce langage nécessite l'installation d'un JRE (Java Runtime Environnement) pour pouvoir fonctionner, qui peut être gourmand en ressources et engendre, selon les machines, des problèmes d'administration supplémentaires.

Le C++ est un langage objet très performant. Cependant, un programme écrit en C++ pose plus de problèmes de portabilité que le C standard, de par les librairies plus spécifiques qu'il utilise ou certaines variantes selon les architectures.

Le C est le langage de prédilection des serveurs. En effet, ce langage est le plus performant car un des plus proches du système (hormis l'assembleur). Il ne propose cependant pas le concept objet, très pratique pour bien structurer son programme et le rendre évolutif.

L'écriture d'une application en ANSI C permet d'assurer qu'il pourra être compilé sur toutes les plateformes supportant le C. De plus, je me rendrai plus tard compte que l'ANSI C est nécessaire pour pouvoir purifier mon code (grâce au logiciel Purify), c'est-à-dire le déboguer dans les moindres détails afin de le rendre le plus fiable que possible.

5.3 Nommage

Afin de ne pas avoir à toujours appeler mon projet « Environnement de Distribution de Tâches sur un Réseau », j'ai décidé de le nommer « Ali » - raccourci d'Alinda, première machine sur laquelle j'ai travaillé, et nom bien en phase avec la nomenclature « Mille et Une Nuits » chère au CDS. Ainsi, je pourrai appeler les différents éléments de l'environnement de cette manière : Sali pour le serveur (Server Ali), Mali pour le moniteur (Monitor Ali), Wali pour l'exécuteur (Worker Ali) et Cali pour le client. Ces dénominations permettent de différencier un serveur d'un autre ; en effet, parler du « serveur » n'est pas toujours parlant puisqu'une multitude d'applications nommées « serveur » existent.

Cependant, par souci de compréhension dans ce document, je continuerai d'appeler les différents éléments d'Ali par leur fonction dans le système.

5.4 Communication inter-démons

Il est évident que la communication entre les différentes composantes d'Ali est une partie importante du projet. Cette communication doit donc être fiable, et simple afin de pouvoir la maintenir plus aisément.

Pour ce qui est de la fiabilité, j'ai décidé d'utiliser le protocole TCP/IP pour toutes les communications ; ce protocole garantit la séquence des données envoyées et leur intégrité. Ainsi, toutes les communications sont faites en mode connecté, c'est-à-dire que deux éléments communicants établissent une liaison statique avant de communiquer ; une fois le ou les messages transmis via cette liaison, elle peut être fermée.

Afin de faciliter la communication inter démons, j'ai créé une surcouche aux fonctions standards d'envoi de données. Ainsi, j'ai créé une structure de message contenant le code du message, le type de données envoyées, la longueur des données et les données elles-mêmes. Cette structure est passée en paramètre à deux principales fonctions de communication : `get_message` et `put_message`, qui respectivement envoient et réceptionnent un message via une *socket* (liaison de communication). Ainsi, ces deux fonctions s'occupent de rendre transparents les problèmes de portabilité, les erreurs de protocole de communication, etc. Le développeur n'a qu'à remplir sa structure message, sans se soucier du reste.

De plus, toutes les communications inter-démons sont régies par des protocoles de communication stricts. Chaque message envoyé est identifié par un code et un type. Les pro-

tocoles de communication spécifient quels codes et quels types doivent être envoyés / reçus, ce qui permet de détecter une erreur dès qu'elle intervient. Tout processus ne respectant pas le protocole sera ignoré (par exemple, un démon externe à Ali qui essayer de se connecter au serveur par mégarde).

5.5 Structure des démons

Chaque démon est constitué d'un processus père qui écoute sur un port donné ; c'est sur ce port que les autres démons pourront le contacter. En règle générale, lorsqu'une connexion arrive sur le port, le processus père se duplique en un fils qui va s'occuper de cette connexion (Figure 6, partie de gauche). Le père reste alors disponible pour l'attente d'autres connexions. Ce procédé est utilisé par le serveur et l'exécuteur ; le moniteur ne se duplique pas, car le traitement de ses requêtes est tellement rapide qu'une duplication est superflue.

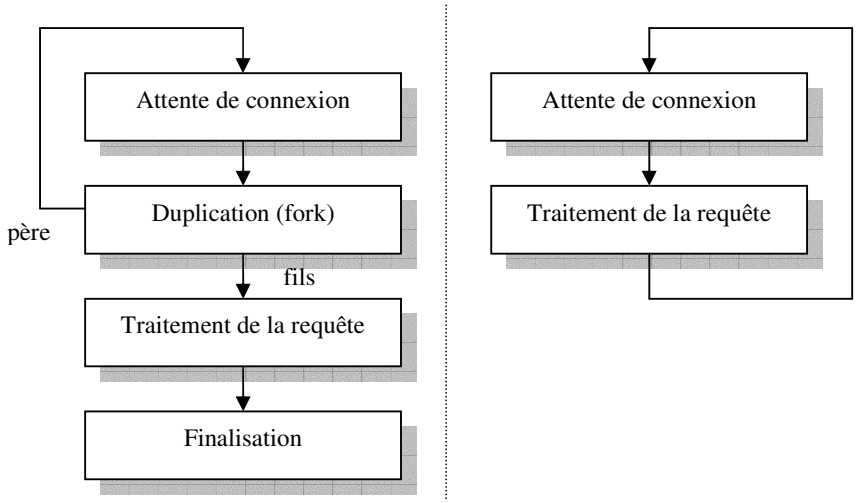

Figure 6. Structure algorithmique générale d'un démon

Cependant, conjointement au processus père du moniteur, un *thread* tourne en parallèle ; ce thread est chargé de la mise à jour des informations sur les exécuteurs. Il tourne en parallèle afin que le service de « demande de meilleur exécuteur » soit toujours disponible, même lors des mises à jour.

Note : les algorithmes plus complets des démons se trouvent dans le Guide Administrateur, placé en annexe.

L'utilisation de la duplication de processus n'est pas forcément inhérente à la création d'un serveur ou d'un démon. En effet, il est aussi possible de simplement créer des threads qui auront le même rôle que les processus fils. Ces threads sont plus légers que les processus, car il n'y a pas création d'un nouveau contexte de processus : ils partagent tous la même zone mémoire, les même descripteurs de fichiers, etc.

Cependant, bon nombre de fonctions standards du C ne sont par réentrantes, c'est-à-dire qu'elles ne peuvent être utilisées que par un seul thread à la fois. Or, Ali est écrit majoritairement avec des fonctions non réentrantes (*hsearch()* par exemple), ce qui rend impossible

l'utilisation des threads de par les conflits mémoire que cela engendrerait. C'est pourquoi tous les démons font de la duplication en processus fils au lieu de la création de threads.

Une autre possibilité serait l'utilisation d'une bibliothèque GNU, la *glibc*, qui règlerait les problèmes de fonctions non réentrantes. De plus, cette bibliothèque propose un bon nombre de fonctions très utiles, telles que les listes, les tables de hachage, etc. Mais mon souci de portabilité et de simplicité à l'installation me force à programmer indépendamment de toute autre bibliothèque que celles qui sont standards.

5.6 Transfert des fichiers

La plupart des requêtes qui sont faites au serveur impliquent des transferts de fichiers. Par exemple, une tâche qui consiste en la compression d'un fichier requiert le transfert de deux fichiers : le fichier à compresser (en entrée) et le fichier compressé (en sortie). C'est pourquoi il faut un moyen fiable pour transférer les fichiers, d'autant plus que se sera souvent l'étape qui prend le plus de temps dans l'exécution d'une tâche.

Plusieurs possibilités s'offrent à nous pour transférer un fichier d'une machine à une autre : l'utilisation d'une partition NFS (Network File System), l'utilisation des commandes de copie à distance (*rcp/scp*) ou alors la réécriture d'un transfert de fichiers fait directement par les démons.

Les partitions NFS sont un très bon moyen pour partager des fichiers entre plusieurs machines sur un réseau. Cependant, l'administration d'un tel système s'avère parfois compliqué, et si un dysfonctionnement intervenait, alors tout l'environnement Ali serait hors service. Ainsi, l'usage de partitions NFS n'est pas recommandé dans ce contexte là.

Les commandes *rcp/scp* semblent être de bons candidats pour le transfert de fichiers ; leur utilisation est simple, puisqu'il suffit de spécifier le fichier de départ et celui de destination. Or, la commande rcp n'est plus installée sur les réseaux fiables, car elle présente de larges failles de sécurité. Il faut utiliser à la place la commande scp, version sécurisée de rcp. Cependant, qui dit sécurité dit cryptage des données ; ce cryptage est long et coûteux lors du transfert des fichiers… comme on l'a déjà dit, la moindre seconde de gagnée lors de l'exécution d'une tâche est un plus pour le service.

Ainsi, j'ai intégré l'envoi des fichiers à la bibliothèque de communication ; un fichier est désormais considéré comme un type de données contenues par un message. Tous les fichiers transférés passent donc « par les démons », ce qui permet une maîtrise complète de ces transferts.

5.7 Description des tâches

Une requête doit pouvoir décrire, de manière plus ou moins précise, le besoin de l'utilisateur, c'est-à-dire la ou les taches qu'il désire exécuter, et comment il souhaite le faire. L'architecture d'Ali implique que ces tâches doivent pouvoir être lancées en arrière plan ; elle ne doivent pas être interactives (c'est-à-dire requérir l'intervention d'un utilisateur, comme par exemple les éditeurs de texte), et n'acceptent en entrée et en sortie que des fichiers. La requête sera généralement stockée dans un fichier, ou dans un buffer (tampon mémoire), avant d'être soumise au serveur via l'interface de soumission. Voici un exemple de fichier de requête :

```
# ------------------------------------------------------------------------------ #
# exemple.req - Fichier de requête exemple.
# Ce fichier de requête donne la description d'une tache qui compresse deux
# fichiers (fich1 et fich2) et les archive par la suite dans fich.tar.
# Les deux compressions sont indépendantes (ce qui est logique), alors que
# l'archivage est dépendant des deux précédentes taches.

%HEADER
%FIFILES fich1 fich2
%FOFILES fich.tar
%SEPARATOR ' '

# Compression du premier fichier
%ID 1
%CMD gzip fich1
%IFILES fich1
%OFILES fich1.gz
%LFILES
%REQUIRE gzip
%DEPS
%COST 1
%TIMEOUT 10
%RETRIES 0

# Compression du second fichier
%ID 2
%CMD gzip fich2.zip
%IFILES fich2
%OFILES fich2.gz
%LFILES
%REQUIRE gzip
%DEPS
%COST 1
%TIMEOUT 10
%RETRIES 0

# Archivage des deux fichiers compresses
%ID 3
%CMD tar -cf fich.tar fich1.gz fich2.gz
%IFILES fich1.gz fich2.gz
%OFILES fich.tar
%LFILES
%REQUIRE tar
%DEPS 1 2
%COST 2
%TIMEOUT 5
%RETRIES 0

# Fin du fichier de requete.
# ------------------------------------------------------------------------------ #
```

Ce fichier est composé de commentaires et de paragraphes constitués de balises. Le premier paragraphe est une entête qui indique quels sont les fichiers qui seront envoyés au serveur (les fichiers d'entrée primaire, FIFILES) et quels sont les fichiers qui sont attendus en sortie (les fichiers de sortie finaux, FOFILES). Chaque paragraphe suivant décrit une tâche à effectuer.

Chaque paragraphe de tâche est décomposé en balises, chacune représentant un paramètre. ID est l'identificateur de la tâche dans la requête et CMD la commande à exécuter. Les IFILES, OFILES et LFILES sont respectivement les fichiers en entrée de la tâche, fichiers en sortie et fichiers liés. Les fichiers liés sont spéciaux, car ce sont des fichiers gérés comme des flux, ce qui permet de faire de l'acquisition de données à la volée. Comme précédemment, je vous renvoie au Guide Utilisateur, disponible en annexe, pour plus de détails. La balise REQUIRE est très importante, car c'est elle qui va permettre l'identification du ou des exécuteurs propices à l'exécution de la tâche donnée. En effet, les applications peuvent être hétérogènes sur le cluster ; par exemple, toutes les machines ne possèdent pas forcément l'application *Marsiaa*. L'insertion d'un « `%REQUIRE Marsiaa` » dans la description de la tâche permet donc de dire : « `ne sélectionner que les exécuteurs disposant de Marsiaa` ».

La balise DEPS est aussi essentielle à la bonne structuration d'une requête. En effet, cette balise permet de gérer les dépendances entre les différentes tâches ; ces dépendances peuvent être soit verticales (3 dépend de 2 qui dépend de 1), soit horizontales (4 dépend de 1, 2 et 3). L'intérêt des dépendances est de pouvoir demander à effectuer des tâches en parallèle, puis de les synchroniser avec une tâche qui nécessite les résultats de ces dernières. Par exemple, le mosaïquage d'Aladin lance 4 tâches de décompression en parallèle, puis une tâche qui soude les 4 images décompressées ; on a donc à faire à une dépendance horizontale.

Les balises COST, TIMEOUT et RETRIES sont des paramètres qui permettent de configurer la tâche. Le coût est utilisé par le serveur pour ses estimations lors de la balance de charge ; il doit être représentatif de la « lourdeur » de la tâche. TIMEOUT permet de définir un temps limite d'exécution, après lequel la tâche est considérée comme échouée. Enfin, RETRIES permet de définir un nombre de tentatives supplémentaires que la tâche peut effectuer lors d'un échec.

Le format d'un fichier de requête est conforme au *Parfile*, format fréquemment utilisé au sein du CDS. De plus, la structure d'une requête est très simple d'utilisation, contrairement à, par exemple, celle d'OpenPBS. En effet, pour une requête de 4 tâches destinées à OpenPBS, il faut écrire un lourd fichier de description assez illisible, en plus de 4 fichiers de scripts qui vont décrire chaque tâche ; pour Ali, un seul fichier de requête suffit.

5.8 Gestion des erreurs

Afin que l'utilisateur ne soit pas perdu lors d'une erreur provenant de lui-même, ou d'Ali (un logiciel n'est jamais à l'abri d'une faille), toutes les erreurs sont remontée via une pile, à l'instar de ce que fait Java automatiquement. En effet, pour chaque fonction sortant en erreur, j'ajoute un message explicatif à une liste d'erreurs. Ainsi, la sortie en erreur peut se propager sur plusieurs niveaux de fonctions, toujours en ajoutant un message explicatif. Ainsi, si une erreur arrive, la requête est annulée et l'utilisateur est notifié de la cause de cette annulation. Les erreurs typiques sont la malformation de la requête, un dysfonctionnement répété d'une des tâches, un problème réseau, etc.

Cette affichage de la pile d'erreurs permet de déboguer beaucoup plus facilement l'application, si par exemple celle-ci connaîtra plus tard (lorsque je ne serai plus là) une erreur que je n'ai pas rencontrée et donc pas traitée.

5.9 Gestion des utilisateurs / sécurité

Concrètement, l'environnement Ali ne gère pas les utilisateurs. En effet, la gestion sécurisée des utilisateurs est une chose compliquée, surtout si on porte l'environnement sur différentes plateformes. De plus, il n'est pas utile de se lancer dans le développement d'une pseudo gestion des utilisateurs, si celle-ci présente la moindre faille de sécurité. Donc, les tâches ne sont pas associées à des utilisateurs, mais seulement à des postes clients (identifiés par leur adresse IP).

Ainsi, le serveur et les exécuteurs sont dotés d'une liste d'adresses IP autorisées. Si un client se connectant n'est pas présent dans cette liste, alors il sera notifié de sa connexion non autorisée et sera ignoré. Il est donc du devoir de l'administrateur d'Ali de n'autoriser que des machines fiables, c'est-à-dire où seuls des utilisateurs dignes de confiance peuvent se connecter.

Cependant, contrairement à OpenPBS, tous les éléments d'Ali tournent sous un compte utilisateur (non root), ce qui permet de restreindre l'accès au sein de la machine hôte. Ainsi, sur un exécuteur, un pirate ne pourrait pirater que les tâches tournant simultanément, mais pas la machine elle-même.

D'autre part, Ali peut très bien être utilisé à travers Internet ; il n'est pas limité à un usage en réseau local. Il faut tout simplement ouvrir les ports utilisés par les démons au niveau du pare-feu, afin que les utilisateurs puissent se connecter. Une fois encore, il faut faire attention aux adresses IP que l'on autorise.

5.10 Purification du code

Une notion très importante lors de la conception d'un logiciel écrit en C est la purification du code. Le C est un langage très proche du système, et la gestion de la mémoire est faite entièrement par le développeur. Cette liberté, permettant au développeur d'écrire des programmes optimisés, contribue très souvent à l'apparition d'erreurs graves (pointeur en dehors d'un tableau, lecture de données non initialisées, écriture sur un bloc mémoire non autorisée, etc.) ; ces erreurs sont souvent vicieuses et difficiles à déboguer. Afin de purifier le code de toutes ces erreurs assez typiques, j'ai été amené à utiliser le logiciel *Purify*, qui indique au développeur chaque opération non autorisée que le programme fait. Cette purification contribue grandement à la fiabilisation de mon système.

L'inconvénient du logiciel Purify, c'est qu'il n'accepte d'analyser que du code écrit en C ANSI ; il n'est dont pas possible d'utiliser la syntaxe de commentaire « // » assez pratique pour commenter une seule ligne, et beaucoup de fonctions courantes ne sont pas reconnues étant donné qu'elles ne sont pas vraiment conformes au C ANSI.

5.11 Affichage des informations

Chaque élément d'Ali génère un certain nombre d'informations qu'il affiche à l'écran. Cet affichage n'est bien sûr pas obligatoire et peut être simplement redirigé vers un fichier ou

/dev/null par l'administrateur lors du lancement. Cependant, il reste très utile pour garder une trace de ce qui se fait à un moment donné.

Le moniteur affiche simplement la liste des exécuteurs qu'il connaît, c'est-à-dire qui sont dans la liste des exécuteurs au démarrage du moniteur, ou alors qui se sont notifiés en cours de route. De base, le moniteur affiche simplement l'état de l'exécuteur - actif (vert) ou non (rouge) -, sa charge, son nom et le nombre de tâches qu'il a effectuées. Cet affichage est personnalisable, pour ainsi montrer plus d'informations permettant de voir plus en détail l'état du cluster.

Figure 7. Moniteur

Le serveur propose un affichage linéaire du déroulement du traitement de la requête. Chaque étape est chronométrée et permet ainsi au programmeur et à l'administrateur de déterminer quelles parties il faudrait potentiellement accélérer. Ainsi, si le temps de transfert des fichiers de résultats est la plus longue des étapes, on peut se demander comment faire pour la rendre plus rapide.

L'affichage d'un exécuteur se présente de la même manière que celui du serveur. Chaque étape est aussi chronométrée pour mieux rendre compte de la répartition des durées.

La coloration du texte en C (sous Unix/Linux) est une fonctionnalité que j'ai découverte durant ce stage, et s'avère très utile lorsqu'on a une multitude de données à afficher… Ainsi, on peut choisir différentes couleurs pour différents types d'informations : jaune pour les temps, vert pour les « warnings », rouge pour les erreurs, bleu pour le déboguage, etc.

Figure 8. Affichage du serveur

Figure 9. Affichage d'un exécuteur.

5.12 Auto-configuration

Je me suis lancé dans la gestion de l'auto-configuration une fois Ali développé. En effet, l'installation de l'environnement était un peu rébarbative, selon les plateformes, car parfois le compilateur n'était pas reconnu, ou mal installé, etc. De plus, il fallait compiler à la main, et deviner d'où provenaient les erreurs si jamais certaines bibliothèques n'étaient pas installées.

Trois outils sont indispensables pour la génération d'un package installable simplement : *autoconf*, *automake* et *autoheader*. Autoconf permet, à partir d'un modèle, de créer un exécutable : *configure*. Celui-ci vérifie si toutes les bibliothèques, compilateurs, variables (etc.) nécessaires au package sont présents. Ainsi, l'utilisateur est notifié de l'absence d'un de ces éléments, et doit se charger de son installation. Automake permet de générer un fichier *Makefile* automatiquement à partir d'un modèle ; ce fichier sera configuré par *configure*, afin de coller à l'architecture actuelle. Enfin, autoheader est un outils qui permet de détecter quelles sont les bibliothèques et fichiers d'entête qui risquent de poser problème dans le package. Il génère un fichier *config.h* qui permet alors de rendre le code portable, en activant ou pas des parcelles de code selon l'architecture. Finalement, pour installer mon package, l'utilisateur ou l'administrateur n'aura à taper que quelques instructions :

```
$ tar –xzvf ali-1.00.tar.gz
$ cd ali-1.00
$ ./configure
$ make
$ make install
```

5.13 Documentation

Dans un projet de cette envergure, un minimum de documentation est nécessaire ; d'une part pour permettre à l'équipe Aladin de maintenir Ali, d'autre part pour expliquer à l'utilisateur comment l'utiliser. J'ai donc réalisé un Guide Utilisateur, qui comme son nom l'indique, est destiné à l'utilisateur de l'environnement : il lui explique comment Ali fonctionne, comment créer une requête, comment la soumettre, etc. Et j'ai aussi écrit un Guide Administrateur, expliquant le fonctionnement d'Ali dans le détail (pour une maintenance et une évolution future), comment l'installer, le configurer, etc.

Tout comme pour l'installation du package, j'ai essayé de réaliser une documentation propre, en s'inspirant de documentations professionnelles, comme celle d'OpenPBS. Ainsi, j'ai écrit ce document en LaTeX, langage que j'ai appris à découvrir et qui est très utilisé dans le milieu des sciences pour son sérieux et sa mise en forme sobre (aussi pour la facilité à éditer des équations, mais cette fonction ne m'est pas utile ici). De plus, j'ai fait relire les documents à plusieurs personnes (dont mes tuteurs), afin de tenir compte d'un maximum de remarques pour coller au mieux aux besoins du lecteur. Enfin, j'ai écrit un mini guide CDS, indiquant à l'équipe quels sont les détails de l'installation actuelle du cluster.

Les Guide Utilisateur et le Guide Administrateur sont disponibles en annexe pour donner plus de détails sur le fonctionnement d'Ali et ses utilisations.

5.14 Quelques statistiques

À titre purement informatif, Ali se résume en 10000 lignes de code, 400h de programmation, 40 pages de documentation et à peu près 100 pauses café.

5.15 Autres utilitaires

Outre tout le système de distribution de tâche, j'ai développé quelques autres utilitaires en rapport avec Ali. Ainsi, j'ai réalisé une interface HTML pour le moniteur (Figure 10), permettant à n'importe qui de connaître l'état du cluster à un moment donné. Ce moniteur affiche les informations nominatives des machines (nom réseau et adresse IP), les charges CPU et réseau, ainsi que la liste des items proposés par les exécuteurs. De plus, la charge CPU est affichée de manière graphique afin d'obtenir rapidement un aperçu de la charge du cluster.

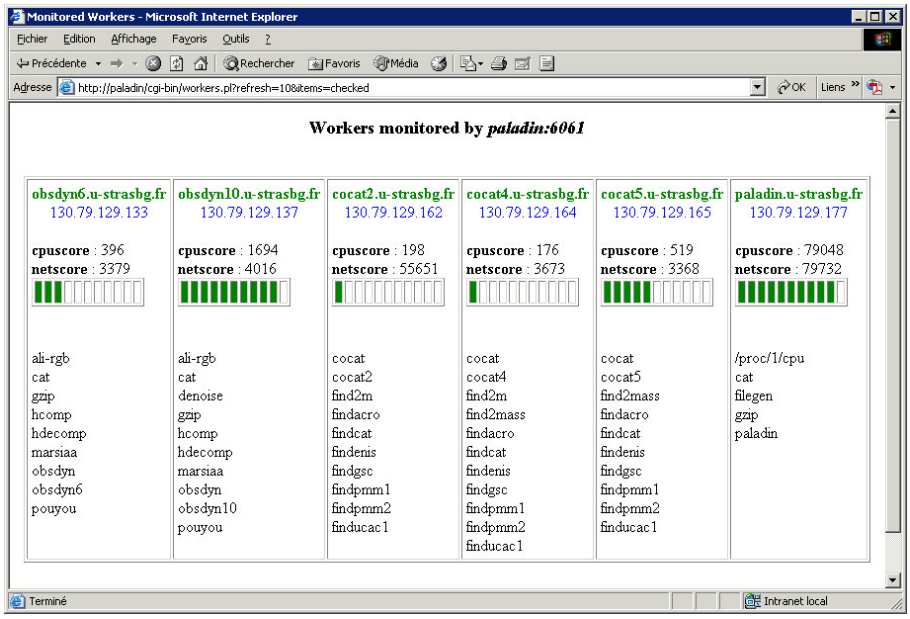

Figure 10. Interface HTML pour le moniteur.

L'intérêt de ce moniteur HTML est qu'il est accessible depuis partout, aucun droit n'est nécessaire pour visualiser l'état du cluster. Étant donné qu'un moniteur est unique au sein d'un même cluster, cette interface était utile lorsque je travaillais en collaboration avec l'équipe VizieR, puisqu'on pouvait alors voir où allaient les tâches, sans nécessairement être le lanceur du démon « moniteur ».

J'ai aussi mis au point un utilitaire Java (Figure 11) permettant de créer des requêtes graphiquement, pour un utilisateur envoyant ses requêtes en mode « manuel ». Cependant, la plupart des utilisateurs d'Ali seront des applications générant de manière automatisée le contenu de la requête, donc ne nécessiteront pas cet utilitaire. Il reste cependant disponible pour ceux qui ne connaissent pas bien la syntaxe du fichier de requête ou qui ne connaissent pas les options de soumission.

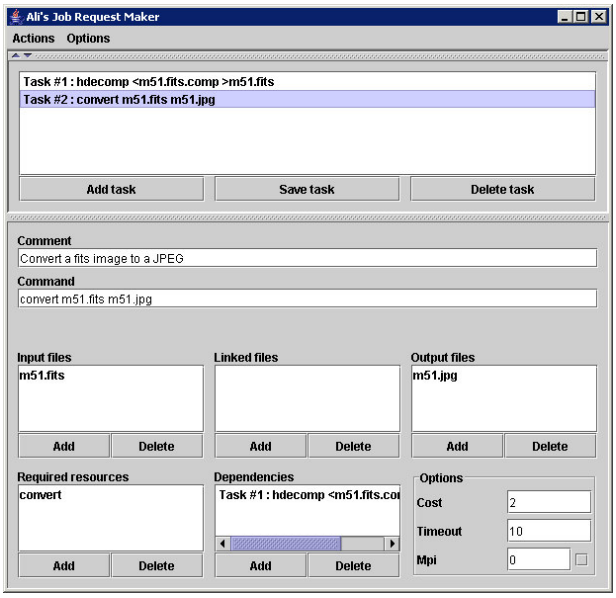

Figure 11. Interface de création de requêtes

6 Exploitation et évolutions

6.1 Mises en exploitation actuelles

Actuellement, déjà deux services en version d'exploitation sont interfacés avec Ali : Marsiaa et la composition RGB d'Aladin. Dans les deux cas, nous avons prévu un mode « roue de secours », c'est-à-dire une exécution du service en local si jamais Ali n'est pas disponible ou si une erreur intervient. Ce mode de secours permet de mettre en exploitation Ali, même si il est encore en phase de développement.

Comme nous l'avons vu, mis à par les temps de calcul des tâches, il faut aussi prendre en compte les temps de transfert des fichiers d'entrée et de sortie. Par exemple, une tâche qui traite un fichier de 10 Mo, et qui s'exécute en une seconde sur Aladin n'est pas une bonne candidate, car il faudrait ajouter au temps d'exécution le coût de transfert des 10 Mo (en plus du fichier de sortie). Chacun des candidats suivants a été choisi en fonction du rapport temps de transfert de fichiers / durée des calculs, afin qu'il soit vraiment utile de les faire passer par Ali.

6.1.1 Marsiaa

Marsiaa est une application qui permet d'effectuer de la segmentation d'images astronomiques grâce à une approche utilisant un modèle Markovien. La segmentation permet d'identifier des zones, selon le rayonnement des objets astronomiques sur plusieurs bandes (différentes longueurs d'onde), comme le montrent les images ci-dessous.

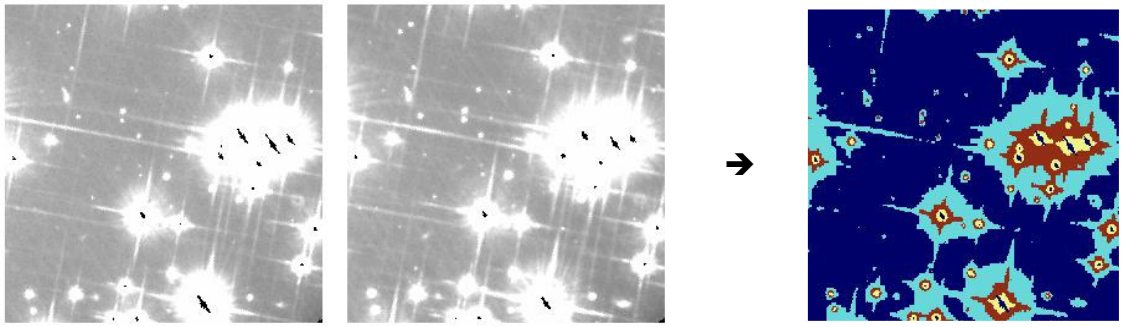

Figure 12. Segmentation de deux images de Hubble prises à 606 et 814 nm

L'interfaçage avec Ali se fait à partir d'une CGI (un service externe à Aladin, mais qui pourrait y être intégré), qui invoque le *submitter* pour envoyer la requête et récupérer les résultats. Le gain en temps pour une requête unique est de l'ordre de 20%, et augmente considérablement si plusieurs utilisateurs demandent une segmentation en même temps. En effet, si plusieurs requêtes arrivent simultanément, chacune sera traitée sur une machine différente, et l'exécution du tout sera donc plus rapide que si chaque requête était traitée sur la même machine.

6.1.2 Composition RGB

AladinJava propose une fonctionnalité qui consiste en la fusion de 3 images de la même partie du ciel, mais à des longueurs d'onde différentes, en une seule image colorée. Cette colorisation permet, d'une part d'avoir des images visuellement plus belles (destinées au grand public), mais aussi de permettre aux astronomes de visualiser simplement quelles sont les différences entres les différentes longueurs d'ondes (chacune représentée par une composante RGB). Dans l'exemple ci-dessous, on peut aisément voir que la Nébuleuse du Crabe est dominée par des gaz rayonnant dans le rouge (émission de rayons infrarouge), rayonne un peu moins dans le bleu et beaucoup moins dans le vert (Rayons X).

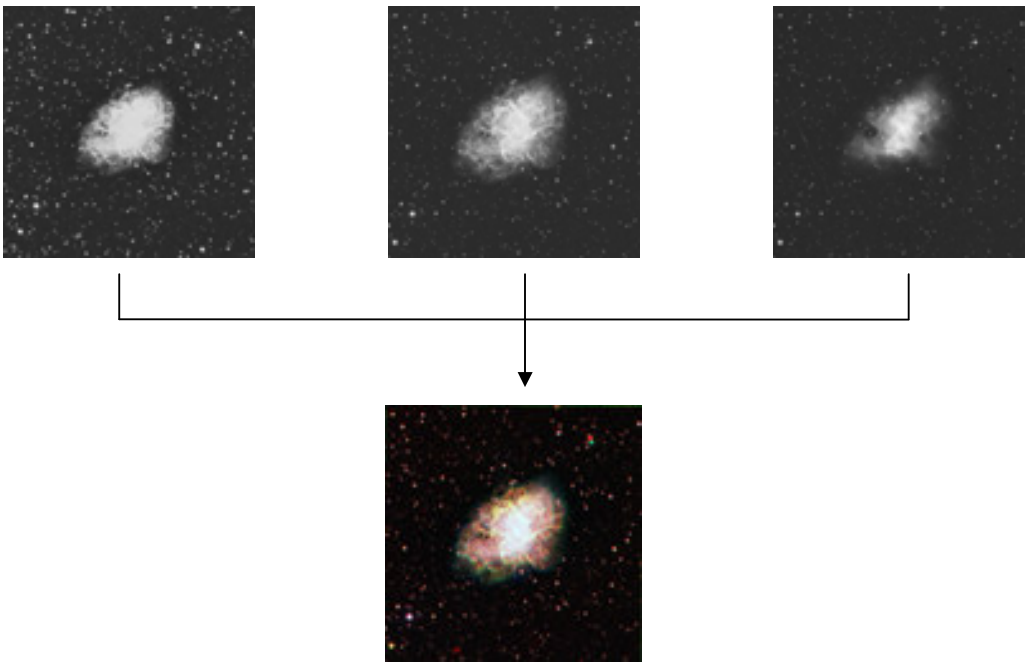

Figure 13. Composition RGB

La composition RGB peut être effectuée en mode non interactif par AladinJava, car celui-ci possède un mode « script » qui accepte une série de commandes à exécuter. Ainsi, la CGI de visualisation de compositions RGB génère un script, qu'elle donne en paramètre dans la requête à Ali. AladinJava, alors installé sur le cluster, va exécuter ce script et renvoyer l'image résultante au serveur Ali, qui la renvoie au client (la CGI de visualisation). Le gain en temps de traitement est de l'ordre de 25% pour une requête unique, et soulage largement le serveur Aladin lors de plusieurs requêtes simultanées. Pour les curieux, ce service de compositions RGB est disponible à l'adresse suivante : http://aladin.u-strasbg.fr/AladinPreview .

6.1.3 Décompressions

Actuellement, le serveur Aladin n'utilise Ali que pour effectuer des décompressions d'images astronomiques. Selon le nombre de requêtes parvenant au serveur, les décompressions peuvent devenir gourmandes, c'est pourquoi il est intéressant de les faire faire par le

cluster. Cependant, pour une requête isolée, il n'est pas vraiment pertinent de l'exécuter via Ali, car le temps de transfert des données est plus long que le temps de traitement de ces données.

6.1.4 Réechantillonage

Les images contenues dans la base de données d'Aladin sont de natures très hétérogène (résolutions différentes, angles de prise de vue différents, balance des gris non identiques, etc.), c'est pourquoi il n'est pas possible de simplement les souder ensemble pour créer une plus grande image. Ainsi, il faut les réechantillonner, c'est-à-dire les ramener à un même « format » (leur donner les mêmes propriétés) ; ce processus prend du temps, peut être effectué indépendamment pour chaque image, donc peut passer par Ali.

6.1.5 Mosaïquage

Le mosaïquage est l'opération qui consiste à déterminer les images « entourant » un point ou un objet demandé dans le ciel, à les décompresser, à les réechantillonner, à les souder et à renvoyer l'image résultante au client. Ce n'est donc pas une opération élémentaire, mais elle peut ainsi constituer une requête contenant beaucoup de tâches à effectuer… c'est cette opération qui à terme devrait être entièrement réalisée par Ali.

6.1.6 Débruitage et déconvolution

D'autres traitements d'image pouvant être assurés par le cluster sont le débruitage et la déconvolution. Le débruitage permet de nettoyer une image astronomique de toutes les interférences qui font apparaître des pixels non informatifs. Ce débruitage peut être fait sur une image découpée, donc peut être parallélisé pour augmenter les performances. Pour l'instant, l'utilisation du débruitage n'est pas utilisable depuis Aladin, mais peut s'effectuer à la main, via Ali.

La déconvolution a pour but de déformer une image, afin de coller au mieux à la réalité. En effet, pour des raisons électroniques, météorologiques et physiques, les capteurs des télescopes déforment l'image réelle. Grâce à un « modèle de déformation », la déconvolution permet de retrouver l'image originale des observations. La déconvolution est gourmande en ressources et non autonome (elle doit être paramétrée pour chaque image de manière manuelle), c'est pourquoi elle n'est pas encore intégrée à Aladin.

6.2 Évolutions à venir

6.2.1 Portage vers Windows

Le portage des démons d'Ali vers une plateforme Windows est une possibilité à envisager. En effet, il se peut que certaines applications de traitement soient écrites exclusivement pour Windows, au quel cas il faudrait installer des machines sous Windows dans le cluster.

De plus, ce serait vraiment une bonne expérience d'hétérogénéité pour l'environnement, et augmenterait son évolutivité.

6.2.2 Communication en UDP

Pour l'instant, toutes les communications sont faites grâce au protocole TCP/IP. À grande échelle, ce protocole peut-être coûteux en ressources système et réseau. Ainsi, pour les messages de petite taille (tels que les mises à jour de la charge CPU des exécuteurs, les ordres d'annulation, etc.), il serait intéressant d'utiliser UDP ; ce protocole ne nécessite pas de connexion active pour envoyer / recevoir des messages. Cependant, il n'assure ni intégrité, ni réception des messages, c'est pourquoi il ne doit être utilisé que pour des messages dont on peut tolérer une perte (si la charge CPU d'un exécuteur n'est pas mises à jour 1 fois sur 100, ce n'est pas très grave).

6.2.3 Augmentation de la taille du cluster

Les 3 machines qui m'ont été mises à disposition ne sont que des machines test, afin de voir si un cluster utilisant mon environnement de distribution est viable ou pas. Finalement, les résultats sont assez concluants, c'est pourquoi le CDS prévoit d'augmenter la taille du cluster, à un nombre de machines pour l'instant indéterminé.

6.2.4 Personnalisation d'un CD de Knoppix

Knoppix est une distribution Linux auto-bootable sur CD, qui ne s'installe pas statiquement ; en fait, le noyau Linux se charge directement en mémoire à partir du CD, très rapidement, et reconnaît la plupart des configurations matérielles. Cette distribution est donc très appréciée pour faire des démonstrations de Linux, rapidement et efficacement.

L'intérêt de Knoppix est que l'on peut personnaliser le CD à notre guise, en modifiant son image et en la regravant. Ainsi, il serait possible de mettre en place Ali sur le CD, afin qu'il s'installe directement sur la machine lors du démarrage de celle-ci. De cette manière, on pourrait créer un cluster de 100 machines en 2 minutes, pour le simple coût des 100 CD (un par machine) ; le réseau est automatiquement configuré via DHCP, donc les exécuteurs n'auraient pas de mal à se connecter au moniteur par défaut. Il serait aussi possible d'envisager une image de Knoppix installée sur un serveur principal, et de créer une disquette de boot afin que la machine aille chercher Knoppix directement sur le serveur ; ceci éviterait de graver un CD par machine, surtout si ce CD est voué à être modifié souvent.

7 Bilan

Pour commencer, il faut bien évidemment se poser la question : « Est-ce que le sujet a été réalisé dans son intégralité ». La réponse, je pense, est « oui » ; une analyse a été faite et il en a été conclu qu'il serait intéressant de développer notre propre environnement de distribution de tâches. Cet environnement a été développé, est fonctionnel et déjà mis en exploitation ; ainsi, les objectifs principaux du stage ont été atteints.

Cependant, ce stage s'inscrit dans un contexte beaucoup plus global : une grille internationale qui permettrait de partager les ressources de milliers d'ordinateurs, supercalculateurs, serveurs de stockage, etc. Avant d'en arriver là, on peut déjà penser à réaliser des applications optimisées pour les clusters : au lieu de faire de la simple distribution de tâches, qui n'accélère pas vraiment une requête isolée, il serait bon d'écrire les applications pour qu'elles puissent tirer parti d'un grand nombre d'ordinateurs (par exemple en faisant du code parallèle écrit avec MPI). Pour l'instant, cette solution n'est pas à l'ordre du jour mais Marsiaa, par exemple, est déjà sur la liste des applications qui seront parallélisées plus tard.

D'autre part, au fur et à mesure que le projet avançait, il s'est avéré qu'Ali n'était pas simplement compatible avec les besoins du CDS. Cet environnement se rapproche beaucoup (conceptuellement, sûrement moins en terme de fiabilité) des solutions professionnelles telles que OpenPBS. L'installation d'Ali sur bon nombre de machines sur Internet pourrait former une mini grille de calcul, entre différents laboratoires par exemple. Pour l'instant, les performances réseau d'Internet sont pénalisantes à cause du temps de transfert des données, mais cette pénalité diminue de plus en plus avec la généralisation des connexions haut débit. Cependant, pour des tâches vraiment axées sur du calcul et ne nécessitant que peu de données (décryptage d'un code, résolution d'équation, rendu d'images 3D complexes), les performances d'Ali sont non négligeables.

Enfin, avec la mise en place du système de disponibilité d'items, Ali peut très bien servir de serveur d'applications. Ainsi, une personne ne possédant pas le compilateur LaTeX peut très bien demander à Ali de compiler ses sources pour elle. Il suffit de créer une tâche nécessitant l'item « LaTeX », d'y joindre les fichiers sources, et 2 secondes plus tard le résultat, un .dvi, est là. Les utilisateurs n'ont donc plus besoin de posséder tous les packages, tous les logiciels ou toutes les librairies pour la conversion, l'édition ou la compilation de fichiers ; Ali le fait pour eux. Il suffit simplement de connaître le nom de l'item voulu, et d'une bonne configuration faite par l'administrateur. Cependant, ce « serveur d'applications » se limite aux applications non interactives et non graphiques… peut-être qu'une évolution est à venir ?

Conclusion

Pour finir, ce stage m'a permis de mener du début jusqu'à la fin un projet qui m'a beaucoup appris. Le réseau est un domaine informatique en évolution constante, qui mérite d'être étudié et exploité. La réalisation d'un environnement de distribution de tâches (sur un réseau) est une expérience très bénéfique pour, à la fois, apprendre à programmer des applications réseau, tenir compte des différentes plateformes disponibles et faire de la gestion de processus. L'architecture d'un tel environnement nous fait trouver des petites astuces, auxquelles on n'aurait jamais pensé auparavant, et qui nous serviront très certainement pour la suite de notre cursus professionnel.

De plus, c'est dans une bonne ambiance que ce stage s'est réalisé, ce qui contribue beaucoup à la satisfaction personnelle de cette expérience professionnelle. Je n'hésiterai donc pas à de nouveau postuler pour un stage dans un laboratoire de recherche, et c'est aussi pourquoi j'ai déjà demandé à travailler au CDS cet été.

Bibliographie

Documents écrits :

Sébastien NICAISSE, « Analyse et mise en œuvre d'un cluster dans le domaine astronomique », *Rapport de stage*, 2003.

Maurice Libes, « Utilisation d'un cluster de calcul openMosix et déploiement à l'aide de LTSP ».

Emmanuel Jeannot, « Calcul distribué : une introduction ».

G.A. Geist, J.A. Kohl, P.M. Papadopoulos, "PVM and MPI : a Comparison of Features", *Arcticle,*1996.

A. Prieto, "The Lightweight Protocol CLIC : Performance of an MPI implementation on CLIC".

Albeaus Bayucan, Casimir Lesiak, etc., "Portable Batch System : Internal Design Specification", *Documentation*, 1999.

William Gropp, Ewing Lusk, "Installtion and User Guide of MPICH, a portable implementation of MPI", *Documentation.*

The LAM/MPI Team, "The LAM/MPI Installation Guide – Version 7.0", *Documentation*, 2003.

The LAM/MPI Team, "The LAM/MPI User Guide – Version 7.0", *Documentation, 2003.*

Sites Internet :

Site du CDS :	http://cdsweb.u-strasbg.fr/
Site de LAM/MPI :	http://www.lam-mpi.org/
Site de PVM :	http://www.csm.ornl.gov/pvm/pvm_home.html
Site du projet openMosix :	http://openmosix.sourceforge.net/
Site d'OpenPBS :	http://www.openpbs.org/
Lien vers le projet GNU Queue :	http://sourceforge.net/projects/queue/
Aide pour LaTeX :	http://www.tldp.org//TeTeX-HOWTO-4.html