N° d'ordre : 2584

DIPLÔME D'ÉTUDES APPROFONDIES

Informatique, Automatique et Productique

Université de Technologie de Belfort-Montbéliard

Cohabilitée avec les Universités
de Franche-Comté, de Bourgogne et l'ENSMM de Besançon

FILIÈRE INFORMATIQUE

MÉMOIRE

Portage d'une application numérique asynchrone MPI
sous le middleware Globus
Soutenu le 12 juillet 2004

Par :
François-Xavier CLEMENT
Préparé au :
Laboratoire Informatique de l'université de Franche-Comté (LIFC)
Equipe Distribution et parallélisme
Sous la responsabilité de :
David Laiymani

Jury de soutenance composé de :

M. Jacques Bahi	**Professeur, Université de Franche-Comté**
M. David Laiymani	**Maître de Conférences, Université de Franche-Comté**
M. Michel Salomon	**Maître de Conférences, Université de Franche-Comté**
M. Abder Koukam	**Professeur, Université Technologique de Belfort Montbéliard**

Institut Universitaire de Technologie de Belfort-Montbéliard
rue Engel Gross
90000 Belfort Cedex France
Téléphone : +33 (0)3 84 58 77 00
Internet : http://lifc.univ-fcomte.fr

Remerciements

Au cours de ce stage, qui sanctionne pour moi le DEA IAP et mon cycle d'ingénieur au sein de l'UTBM, j'ai eu le soutien et l'aide de plusieurs personnes que je souhaite remercier, en particulier :

David LAIYMANI, mon responsable de stage, pour ses conseils, son encadrement et sa confiance.

Fabien HANTZ, doctorant au LIFC, pour son aide précieuse au début de mon stage.

Borja SOTOMAYOR, professeur assistant à l'université de Deusto (Espagne), pour ses conseils ainsi que pour son tutorial.

Raphaël COUTURIER, maître de conférences au LIFC, pour ses conseils et ses explications tout au long de mon stage.

Philippe VUILLEMIN, doctorant au LIFC, pour son aide.

ainsi que mon entourage pour leur compréhension et tous les membres de l'équipe Algorithmique Numérique Distribuée (AND) de Belfort pour leur accueil chaleureux.

Sommaire

Table des figures

Liste des tableaux

Table des listings

Introduction

De nombreuses applications scientifiques, telles que le décodage d'une portion d'ADN en biologie par exemple, sont par nature très gourmandes en ressources de calcul [Fos03]. Or, depuis quelques années les performances des ordinateurs personnels évoluent à une vitesse impressionante [Fos02a]. Ce phénomène, associé à des prix attractifs, nous permet d'envisager la mise en parallèle d'un grand nombre de ces machines plutôt que d'investir dans des ordinateurs multiprocesseurs. Ainsi, le réseau mis en place pourra être vu comme une machine parallèle *virtuelle* (on parlera également de Grappe ou cluster).

L'avènement de l'internet, nous permet de voir plus loin et de penser à des ressources « virtuellement illimitées ». L'accès à ces ressources restant transparent, on parle alors de *grille* [Fos02b, Fos02c, Fos00, Sag]. C'est ainsi que depuis quelques années une thématique scientifique a vue le jour : le *calcul sur grille* (ou *Grid Computing*) [IF99, Rom03]. Ainsi, de nombreux systèmes tels que Legion [NNTHG02], Globus [FGNC01, ea02, FK99] et XtremWeb [FGNC01] ont alors émergés, fournissant les mécanismes de bases aux grilles de calcul (sécurité, gestion de ressources,...).

Cependant, le concept de grille de calcul a fait resurgir des problèmes qui avaient disparus avec l'utilisation des clusters ou des grappes. En effet, le developpement d'applications destinées à être exécutées sur une machine parallèle virtuelle demande un environnement de programmation particulier où les temps plus ou moins long que mettront les informations à circuler entre les processeurs devront être pris en compte (volume d'information échangé et encombrement du réseau). Or, pour une application de calcul sur grille ce paramètre de temps de communication devient d'autant plus important que les distances à parcourir sont grandes. Ainsi le problème de l'hétérogéneité des connexions réseau et des puissances de CPU devient primordial dans le contexte de calcul sur grille.

Dans le cadre du calcul parallèle, les solutions les plus simples consistent à synchroniser, au cours des itérations, les unités impliquées dans le calcul. Dans ce cas, les communications sont fortement pénalisantes pour les performances globales de l'application car elles entraînent souvent des périodes d'inactivité pour les processeurs concernés. Une autre solution consiste à désynchroniser les itérations, permettant ainsi de trouver une bonne approximation du résultat recherché après avoir vérifié les propriétés garantissant la convergence. Ces méthodes sont appelées les *méthodes itératives asynchrones* [BT89]. Elles permettent de réduire significativement les temps de calcul par rapport aux méthodes synchrones, car il n'y a pas de « temps morts » dus aux synchronisations entre les itérations.

De part cet asynchronisme de calcul et de communication, ces méthodes itératives

semblent particulièrement bien adaptées au calcul sur grille. En particuler dans [JBV04], Bahi *et al* montrent l'efficacité de telles méthodes en utilisant l'environnement MPI (Message Passing Interface). Ce dernier n'étant pas un environnement de calcul sur grille à proprement parlé. **Le travail réalisé au cours de ce stage avait pour objectif d'étudier le comportement d'une application asynchrone dans un environnement de calcul sur grille (Globus) et de comparer les résultats obtenus avec ceux de la même application réalisée sous MPI [BDV94, SL03]**.

La première partie de ce rapport présente rapidement le laboratoire où j'ai effectué mon stage. La seconde, quant à elle, sera consacrée à une étude bibliographique, dans laquelle nous présentons le concept de grille de calcul et le middleware Globus. Enfin la troisième et dernière partie décrira l'application asynchrone choisie, ainsi que son portage de MPI vers GLOBUS. Nous détaillerons également dans cette partie les résultats obtenus lors des divers tests effectués, résultats plus qu'encouragents quant à l'utilisation de Globus comme support d'exécution des méthodes itératives asynchrones.

Chapitre 1

Le Laboratoire d'Informatique de l'université de Franche-Comté (LIFC)

Dans cette première partie, nous allons tout d'abord présenter le laboratoire du LIFC[1] au sein duquel j'ai effectué mon stage. Ce dernier comptant à la fois pour le DEA IAP[2] et pour mon projet de fin d'étude validant mon cursus au sein de l'UTBM[3].

Ensuite, nous présenterons, plus particulièrement, l'équipe DP « Distribution et parallélisme », équipe dans laquelle j'ai evolué durant ces six mois de stage.

1.1 Présentation

Depuis sa création, le laboratoire d'informatique n'a pas cessé de s'agrandir et de se développer. Pour preuve, le budget de l'année 2003 est six fois supérieur au budget de l'année 1998. Les quatre dernières années ont vu ce mouvement s'accélérer et depuis 2003, le laboratoire est reconnu C.N.R.S. (FRE 2661).

En 2000, le laboratoire d'informatique, qui s'appelait le Laboratoire d'Informatique de Besançon (LIB), est devenu le Laboratoire d'Informatique de l'Université de Franche-Comté (LIFC) pour prendre en compte le fait que la moitié des enseignant-chercheurs en informatique de l'Université de Franche-Comté est en poste sur le site de Belfort-Montbéliard.

Le développement rapide et cette évolution multi-sites a été l'occasion d'entamer une réflexion en profondeur sur les structures du laboratoire. La direction est constituée d'un Directeur et de deux Directeurs adjoints. Au moins une des trois personnes qui occupent ces fonctions est sur le site de Belfort-Montbéliard. L'équipe de direction est assistée d'un bureau de direction qui réunit l'ensemble des directeurs de recherches (professeurs, MCF

[1] **LIFC** : Laboratoire d'Informatique de l'université de Franche-Comté
[2] **IAP** : Informatique Automatique et Productique
[3] **UTBM** : Université Technologique de Belfort-Montbéliard

HDR,...).

Enfin, un conseil de laboratoire, composé des représentants de chaque catégorie de personnel (MCF, Doctorants, IATOS,...) examine trois fois par an les principales propositions de l'équipe de direction et conseille celle-ci.

1.2 Personnel et répartition géographique

Grâce à un développement des formations en informatique à l'UFR Sciences et Techniques sur le site de Besançon, à l'IUT de Belfort-Montbéliard et à l'UFR Sciences et Techniques de Gestion Industrielle sur le pôle universitaire de Montbéliard, les effectifs du laboratoire ont augmenté régulièrement. Les recherches qui étaient dirigées par six professeurs en 1997, le sont par dix en 2004. Le groupe d'enseignant-chercheurs installé sur le site de Belfort-Montbéliard, qui était constitué d'un MCF en 1996, est constitué aujourd'hui de deux professeurs et seize MCF. Enfin, depuis 1997, le laboratoire a appliqué une politique de recrutement d'enseignant-chercheurs largement ouverte sur l'extérieur. Parmi les dix-sept enseignant-chercheurs recrutés en quatre ans, dix sont extérieurs, venant de l'ENS Cachan, Grenoble, l'ENS Lyon, Marseille, Toulouse, Nancy et Versailles.

Actuellement le laboratoire compte :
- 10 Professeurs,
- 29 Maîtres de Conférences,
- 33 Etudiant-chercheurs (doctorants),
- 9 Ingénieurs contractuels.

1.3 Résultats obtenus durant la période 2000 - 2003

Les résultats obtenus ces dernières années sont les suivants :
- Labellisation CNRS en 2003,
- Labellisation d'un projet INRIA en partenariat avec le LORIA en 2003,
- Participation à 5 projets RNTL, 1 projet RNRT et 1 projet cognitique,
- 5 contrats de recherche (AID, Schlumberger, PSA, CEA, CNET),
- 3 habilitations à diriger des recherches,
- 17 thèses soutenues,
- 107 communications internationales et nationales,
- 44 articles en revues internationales et nationales,
- Réalisation d'un projet d'innovation ANVAR,
- Participation à un projet Européen ITEA,
- Participation à 2 projets Interreg III,
- Participation à une ACI jeunes chercheurs,
- Leader d'une ACI Sécurité Informatique et participation à 2 autres.

1.4 Insertion dans un pôle de recherche régional

En 1998, le laboratoire créait l'ISTI (Institut des Sciences et Technologies de l'Information en Franche-Comté) avec deux partenaires : le LAB (Laboratoire d'Automatique de Besançon) et MTI@SHS (fédération d'enseignant-chercheurs en Sciences Humaines et Sociales appartenant à plusieurs laboratoires comme THEMA, Chrono-Ecologie, LASELDI). NUMERICA (Centre de Développement du Multimédia de Franche-Comté à Montbéliard) et le CLA (Centre de Linguistique Appliquée) sont associés à l'ISTI depuis 1999.

1.5 Les collaborations nationales et internationales

Le LIFC bénéficie de nombreuses collaborations avec d'autres structures de recherche en France et dans le monde :

Collaborations Nationales :

- I3S, LAB, LIP, LSR-IMAG, LORIA dans le cadre des projets RNTL[4] et RNRT[5],
- Université de Nancy 2 dans le cadre du projet cognitique ICOGAD,
- GDR ALP, participation aux groupes LODEC et MELANGES,
- GDR ARP, animation du groupe RGE, participation au groupe Grappes,
- GDR I3,
- France Telecom R&D, co-encadrement de 3 thèses.

Collaborations Internationales :

- Oregon Graduate Institute, USA, thèse en mobilité
- ENSIAS, Maroc, action intégrée
- University Sains Malaysia, Malaisie, co-encadrement de thèse
- Université de Westminster, Grande-Bretagne, co-encadrement de thèse
- Open University, Grande-Bretagne, projet cognitique ICOGAD
- Université de Waikato, Nouvelle Zélande, projet BZ-Testing-Tool
- UUNET worldcom, USA, thèse en entreprise

1.6 Présentation de l'équipe DP « Distribution et parallélisme »

L'équipe DP oriente ses travaux vers deux axes stratégiques :

- Métacomputing et Algorithmique numérique,
- Téléapplications et multimédia mobile

Ces axes de recherche s'appuient sur les compétences acquises durant les années précédentes (1998-2001), et prennent en compte d'une part, les développements technologiques récents en matière de réseau et de terminaux et d'autre part, les nouveaux besoins industriels.

En effet, avec l'apparition des communications mobiles de troisième génération, la mise en place de backbones hauts débits dans le monde, la montée en puissance des clusters de

[4] **RNTL** : Réseau National des Technologies Logicielles

[5] **RNRT** : Réseau National de Recherches en Télécommunications

stations et l'émergence de nouveaux terminaux portables et sans fil, de nouvelles applications irréalisables il y a quelques années deviennent possibles et vont révolutionner notre mode de travail.

L'axe **algorithmique numérique et métacomputing** regroupe ses projets dans le domaine des grilles de calcul et l'algorithmique haute performance. Il est composé de deux actions :

- L'action **métacomputing** intègre les travaux sur la mise en place d'infrastructures nécessaires à l'exécution de calculs sur la grille et, d'une manière plus générale, dans les environnements de réseaux étendus,
- L'action **algorithmique numérique distribuée** prend en compte la résolution de problèmes scientifiques par des algorithmes itératifs et la mise en place d'outils adaptés permettant d'améliorer l'exécution de ces applications.

Ces deux actions sont fortement liées entre elles sur trois points : la distribution à grande échelle, l'allocation de ressources et les applications hautes performances. Les travaux effectués dans ces domaines sont donc réalisés en concertation dans le but de développer une expertise commune.

C'est au sein de l'équipe Algorithmique Numérique Distribuée que j'ai effectué mes 6 mois de stage de fin d'études.

Chapitre 2

Les grilles de calcul

Dans cette partie sont présentées les caractéristiques des grilles de calcul. Nous décrivons brièvement le contexte actuel, le concept des grilles de calcul, puis leurs contraintes et enfin leurs différentes composantes.

2.1 Contexte actuel

Le concept de « grille de calcul » (Grid) est apparu suite à la demande des scientifiques de disposer d'une très grande puissance de calcul et d'une capacité de stockage de l'ordre du Péta Octet [HJN01]. En effet, dans de nombreux domaines, le travail des chercheurs nécessite l'accès et l'échange d'importantes bases de données à travers l'Internet, afin de réunir et de comparer entre elles leurs informations, d'établir des statistiques, d'analyser les résultats...

Ainsi, de nombreux problèmes de modélisation par exemple correspondent à une utilisation intensive de ressources de calcul mais seulement pendant un laps de temps réduit. L'investissement nécessaire ne peut être pris en charge par des entités isolées, mais exige la mise en place de collaborations plus larges, au niveau international.

Les centres de calcul en tant que tels ne suffiront plus d'ici quelques années. Le concept de grille de calcul sera sans doute une réponse à cette demande.

2.2 Concept

Le concept de base des Grilles [Cha03] reprend celui des fournisseurs d'électricité [FK00, Duc03]. À savoir, offrir l'énergie nécessaire à un client pour faire fonctionner un appareil d'une puissance donnée à un instant donné sans se soucier des contraintes inhérantes (puissance, source, etc). Ce concept a été étendu et repris dans la perspective de fournir des ressources informatiques.

L'objectif à terme des grilles est de pouvoir consommer de la puissance de calcul, comme on consomme du courant électrique, sans connaître nécessairement le fournisseur. Pour cela, cette technologie permet de mettre en liaison un très grand nombre de ressources (plusieurs millions), afin de créer des « organisations virtuelles » [FKT01].

Une telle grille suppose bien sûr une infrastructure réseau sous-jacente suffisamment

performante [HJN01]. L'evolution des technologies de communication rend aujourd'hui disponible des réseaux à très hauts débits sur de longues distances, alors que jusqu'à présent les hauts débits étaient réservés aux réseaux locaux.
Il est généralement reconnu que les domaines d'application des « grilles de calcul » se divisent essentiellement en cinq grandes classes :

1. Le calcul intensif distribué, où l'on utilise les grilles de calcul pour additionner les capacités de plusieurs machines inefficaces individuellement afin de résoudre un problème donné. Ce peut être l'agrégation de plusieurs super-calculateurs ou simplement de stations de travail. Les applications typiques sont du domaine de la simulation, que ce soit de situations complexes (manoeuvre militaires par exemple) ou du déroulement précis de processus physiques.
2. Le calcul à haut débit où l'on utilise les grilles de calcul pour ordonnancer un grand nombre de tâches, peu ou pas du tout couplées, sur des machines inutilisées. Comme dans le cas du calcul intensif distribué, le résultat peut être focalisé sur la résolution d'un problème unique, mais la nature indépendante des tâches impliquées fait que l'on s'intéresse à des problèmes très différents. Le constructeur de microprocesseurs AMD a utilisé cette technique lors de la conception de ses modèles K6 et K7.
3. Le calcul à la demande, qui permet une utilisation temporaire de ressources dont la possession permanente ne serait pas rentable : capacité de calcul, logiciels, bases de données, ... Ici c'est le rapport qualité/prix qui est l'argument principal, plutôt que les performances absolues.
4. Le traitement intensif de données, où l'on produit de nouvelles informations à partir de données géographiquement distribuées. C'est le cas typique de la Physique des Particules à l'échéance du LHC (2006), des systèmes de prévisions météorologiques, des programmes d'observations terrestre,...
5. Le calcul collaboratif, qui privilégie la mise en place d'interactions entre personnes humaines pour l'exploration conjointe de bases de données, des systèmes de réalité virtuelle à objectifs éducatifs ou de distraction...

Les utilisateurs potentiels d'une grille de calcul couvrent un très large éventail de la société. Cela peut aller d'un gouvernement voulant simuler des situations de crise ou simplement fournir une grande capacité de calcul à la recherche fondamentale, jusqu'à la fourniture en ligne de jeux interactifs. Un nombre incalculable d'applications peut être envisagé dans les domaines de la santé publique, de la finance,... Des PME-PMI pourront ainsi avoir accès à des outils informatiques et à des banques de données qui leur étaient autrefois interdits du fait même du coût des moyens requis.

2.3 Contraintes

Interconnecter un ensemble de machines hétérogènes nécessite de prendre en compte une multitude de paramètres :

- la sécurité ;
- l'authentification des utilisateurs ;
- la gestion des données ;
- la migration de données entre machines ;
- l'identification des ressources disponibles pour l'exécution d'un programme.

2.4 L'architecture d'une grille de calcul

L'objectif ici n'est pas de décrire une grille de calcul de manière exhaustive, mais d'en présenter les composantes principales. Cette architecture peut être découpée en différentes couches, comme le montre la figure 2.1. Une couche est une abstraction représentant un ensemble de fonctions du système qui permettent des actions de même niveau [Cha03]. Chaque couche fait appel aux services de n'importe quelle couche inférieure.

À la base du modèle, la **couche Fabrique** représente le niveau le plus proche des ressources « physiques ». Il s'agit de l'ensemble des serveurs, le stockage et le réseau qui fournissent les ressources nécessaires aux applications.

La **couche Connectivité** représente les protocoles fondamentaux régissant la communication (échange de données entre les ressources) et l'authentification (d'un point de vue sécuritaire, identification des ressources et des utilisateurs, ainsi qu'une sécurisation des échanges).

La **couche Ressource** représente le partage de ressources individuelles. C'est elle qui a notamment en charge la mise à disposition et la facturation à l'utilisation. Elle utilise les services de communication et de sécurité et permet de construire le protocole sous forme d'applications de deux services sécurisés :

- service information : sur les ressources partagées et leur état ;
- service gestion : gérer l'accès aux ressources partagées (allocation, accès et droit, qualité de service, monitoring, contrôle,...).

La **couche Collectif** se charge de la coordination des ressources. Elle comprend les services d'annuaires, la prise en charge des demandes, la surveillance et la réplication des données. C'est l'orchestrateur des ressources.

Enfin, la **couche Application** représente l'ensemble des fonctions qui ont été développées pour interagir avec la grille.

Nous allons maintenant présenter de manière plus concrète le rôle des cinq couches qui viennent d'être citées.

2.4.1 La couche Fabrique

La couche Fabrique [Cha03] « fournit » les ressources physiques telles que des processeurs pour le calcul, du stokage, des bases de données, des annuaires ou des ressources réseau. Une ressource peut également être une entité logique comme un système de fichiers distribué, ou un serveur virtuel dans le cas d'un *cluster* d'ordinateurs. La couche Fabrique est la plus basse du modèle et elle est en relation directe avec le matériel pour mettre à disposition les ressources partagées. Lorsqu'une demande d'accès à une ressource est formulée, par le biais d'une opération de partage d'un niveau supérieur, un composant logiciel du niveau Fabrique est invoqué. Le rôle de ce composant est d'agir directement sur les ressources logiques ou physiques de la grille.

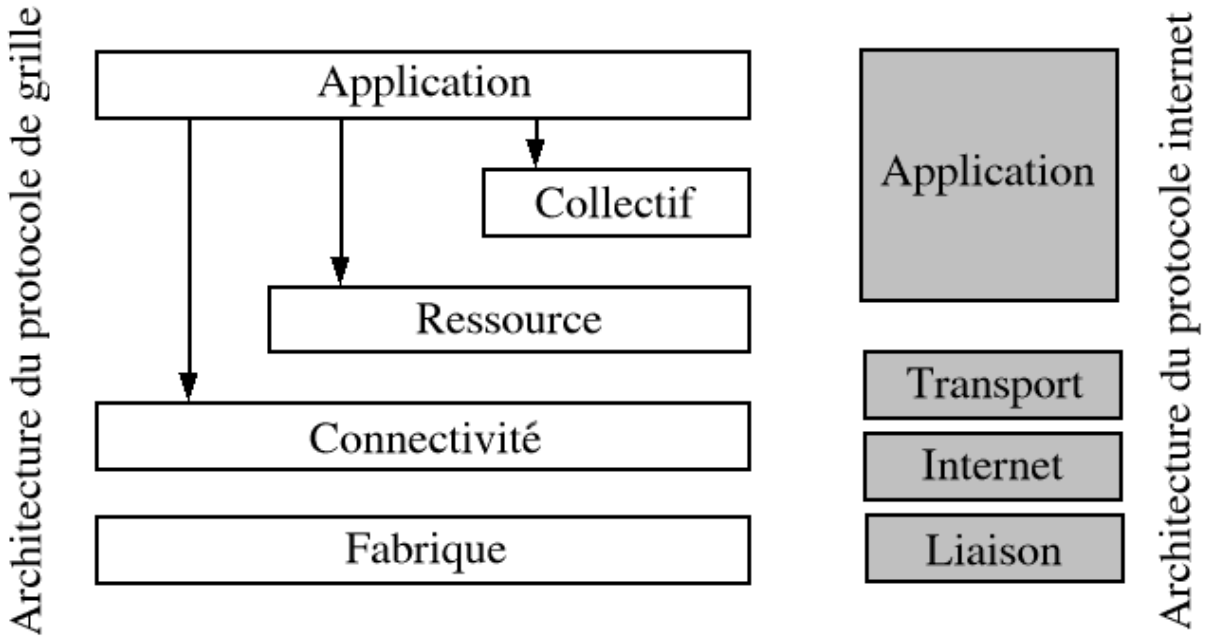

FIG. 2.1 – *L'architecture d'une grille de calcul et sa comparaison à celle du protocol Internet.*

2.4.2 La couche Connectivité

La couche Connectivité [Cha03] implémente les principaux protocoles de communication et d'authentification nécessaires aux transactions sur un réseau de type grille. Les protocoles de communication permettent l'échange des données à travers les ressources du niveau fabrique. Les protocoles d'authentification s'appuient, quant à eux, sur les services de communication pour fournir des mécanismes sécurisés de vérification de l'identité des utilisateurs et des ressources.

L'architecture d'une grille de calculs est parfois représentée sous une forme de sablier (voir fig. 2.2). La métaphore du goulet d'étranglement au niveau connectivité permet d'imager l'idée de la limitation du nombre de protocoles utilisés (protocoles de communications et protocoles réseau) pour faciliter le partage des ressources. À l'inverse, la base et le haut du sablier (les parties élargies) symbolisent la diversité des ressources et des applications.

L'utilisation de protocoles standards simplifie également le développement et le déploiement des solutions. Ainsi, pour la communication et la sécurité, de nombreux protocoles développés pour Internet ont été réutilisés. Pour la communication, les protocoles nécessaires sont ceux relatifs au transport, au routage et à la gestion des noms. Le choix se porte essentiellement sur les protocoles de la pile TCP/IP. TCP et UDP sont utilisés pour le transport, IP pour le routage, ICMP pour la surveillance, et DNS, par exemple, pour les applications.

En dehors des aspects de communication, la couche connectivité a en charge la majeure

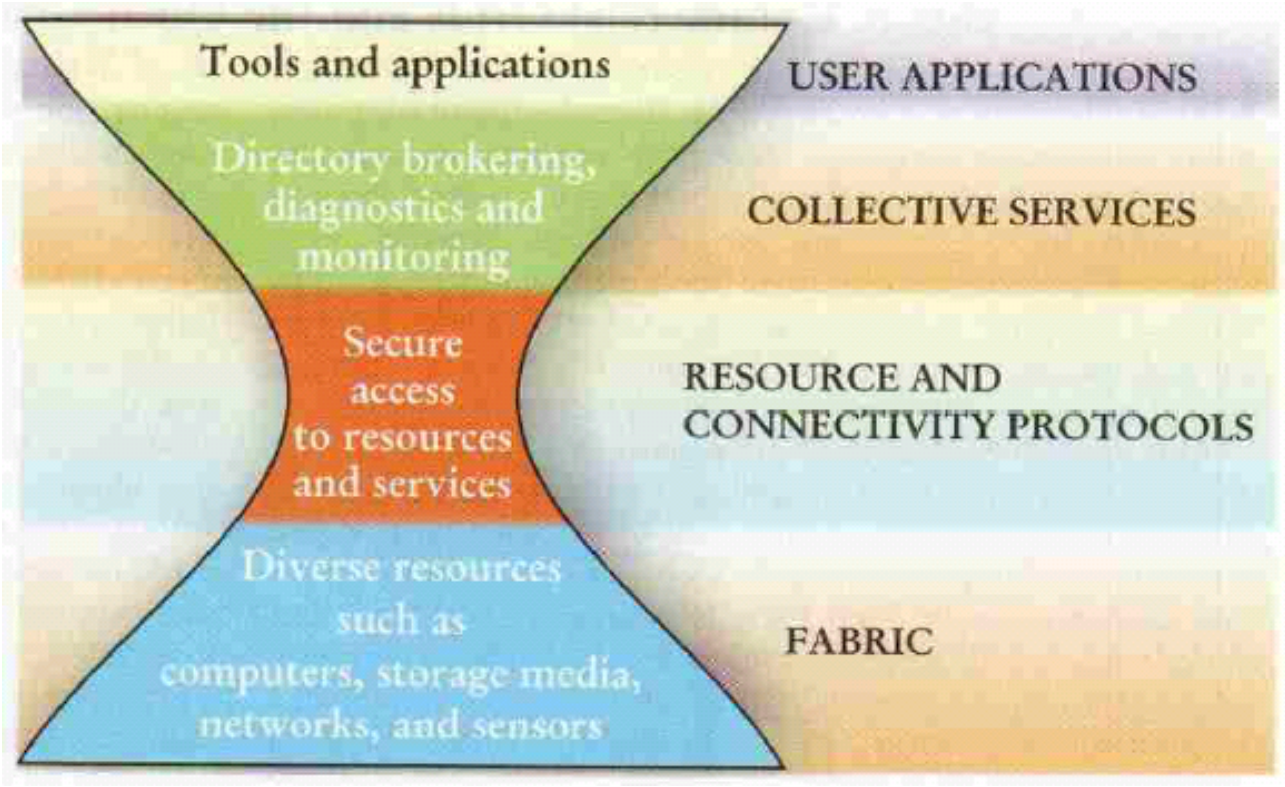

FIG. 2.2 – *L'image du sablier.*

partie de la sécurité du système. Pour ce faire, les concepteurs de la grille ont également recours autant que possible, aux protocoles existants. Toutefois les besoins en matière de sécurité ne doivent pas rendre l'utilisation des grilles trop contraignante. Le cahier des charges relatifs à la sécurité et au fonctionnement d'une grille comporte quatre points essentiels :

- La necessité pour les utilisateurs de s'authentifier une fois pour toute sur le réseau (« **single sign on** »).
- Un système de « **délégation** » doit permettre à un programme de disposer des permissions de l'utilisateur qui l'a lancé pour pouvoir accéder aux ressources nécessaires, et le cas échéant, que ces permissions soient transmises aux composants logiciels appelés par le programme principal.
- La **sécurité** doit être intégrée au travers des diverses solutions mises en oeuvre sur chaque site distant. Chaque site ou fournisseur de ressources dispose de ses propres outils de sécurité. La sécurité au niveau de la grille doit prendre en compte chacune de ces spécificités et mettre en oeuvre des solutions permettant de s'adapter aux contextes locaux.
- La mise en oeuvre de **relations de confiance** (relations d'approbations) entre les sites doit être prévue. De cette façon, si un utilisateur fait appel à deux ressources de sites différents, et s'il n'est accrédité qu'auprès d'un seul site, le second devra l'accepter en faisant confiance aux politiques de sécurité du premier.

2.4.3 La couche Ressource

La couche Ressource [Cha03] utilise les services des couches Connectivité et Fabrique pour collecter les informations sur les caractéristiques des ressources, les surveiller et les contrôler. Elle s'occupe également de l'aspect facturation. La couche Ressource ne se préoccupe pas des ressources d'un point de vue global, elle ne s'interesse pas à leur interaction. Ceci incombe à la couche Collectif abordée dans la section suivante. La couche Ressource ne s'intéresse qu'aux caractéristiques intrinsèques des ressources et à la façon dont elles se comportent.

Au niveau Ressource, deux types de protocoles sont à distinguer. D'une part, sont utlilisés des protocoles permettant d'obtenir des informations statiques sur la structure d'une ressource telles que sa configuration, ses stratégies de sécurité, ses coûts d'utilisation, ou permettant d'obtenir des informations dynamiques telle que la charge de travail à un instant donné. D'autre part, les protocoles de gestion du niveau Ressource ont en charge la **négociation pour l'accès aux ressources partagées**. On peut ainsi spécifier des conditions d'accès - la réservation par exemple - ou demander à ce que soient annoncées à l'avance les opérations qui seront effectuées, comme la création de processus ou les accès aux données, par exemple. Cette couche a en charge le *monitoring* (surveillance et remontée des alarmes) des opérations. Elle contrôle l'exécution des traitements et signal les erreurs rencontrées aux couches de niveaux supérieurs qui souhaiteraient en être informées. Ainsi, les protocoles de gestion mis en oeuvre à ce niveau sont responsables du bon usage des ressources, en liaison avec les stratégies définies par le concepteur de la grille pour l'utilisation des ressources partagées.

2.4.4 La couche Collectif

La couche Collectif [Cha03] se charge des interactions entre les ressources. Elle gère l'ordonnancement et la co-allocation des ressources en cas de demande des utilisateurs faisant appel à plusieurs ressources simultanément. C'est elle qui choisit sur quelle ressource de calcul faire exécuter un traitement en fonction de coûts qu'elle sait estimer. Elle s'occupe également des services de réplication des données. En outre, elle a en charge la surveillance des services et elle doit assumer la détection des pannes. En un mot, elle a un rôle « d'orchestrateur » de l'ensemble des ressources du système.

Un des rôles essentiels au fonctionnement du système que joue cette couche est le rôle d'**annuaire**. Un annuaire est une base de données répertoriant les ressources et leurs différentes caractéristiques. Quand une application va devoir utiliser une ressource, un « courtier de ressource » (*Resource Broker*) va chercher dans l'annuaire celle correspondant le mieux aux exigences requises par la tâche. Une fois que l'annuaire aura transmis la localisation de la ressource, la tâche pourra s'exécuter.

2.4.5 La couche Application

La couche la plus haute du modèle est la couche Application [Cha03] qui correspond aux logiciels qui utilisent la grille pour fournir aux utilisateurs ce dont ils ont besoin, qu'il s'agisse de calcul ou de données. Les applications utilisent des services de chacune des

couches de l'architecture. Les **couches Collectif et Ressource** sont sollicitées pour la recherche des ressources. Après les avoir trouvées, puis s'être authentifiées au travers de la **couche Connectivité**, les applications utilisent les services du **niveau Fabrique** pour y accéder.

Dans le chapitre suivant, nous décrivons un environnement de grille de calcul se basant sur les concepts énoncés précédement : le projet « Globus ».

Chapitre 3

Le middleware Globus

Dans ce chapitre nous présentons le Projet Globus [Sou03]. C'est un projet open source visant à créer les logiciels et les outils nécessaires pour la conception et la mise en oeuvre de grilles de calcul. Globus est principalement développé aux Etats-Unis, au sein de l'Argonne National Laboratory par l'équipe de Ian Foster. Le travail sur Globus a commencé en 1997 et le projet est toujours actif.

Le « Globus Toolkit » est formé d'un ensemble de composants. Son architecture modulaire permet d'apporter les modifications et les améliorations d'une manière rapide et efficace. Globus est devenu le standard ipso facto utilisé dans les projets de grilles de calcul. De nombreuses entreprises ont ainsi adopté Globus comme brique de base à leurs produits commerciaux de grilles de calcul. Dans la suite de ce chapitre, nous allons présenter les principales fonctionnalités offertes par Globus à ses utilisateurs en terme de sécurité, de services d'information, de gestion des communications, de gestion des ressources et de traitement des données.

3.1 Introduction

Globus [Sou03] fournit les fonctionnalités et les services de base nécessaires à la construction de grilles de calcul. Ainsi nous trouvons des services et des mécanismes tels que la sécurité, la localisation et la gestion des ressources, la communication...

Il est ainsi composé d'un ensemble de modules ayant chacun une interface que les services de niveau supérieur pourront utiliser pour invoquer ses mécanismes. Nous trouvons ainsi des modules pour [FK97] :

- **Localisation et allocation des ressources** : ce composant permet aux applications d'exprimer leurs besoins en ressources et fournit les mécanismes permettant d'identifier les ressources adéquates.
- **Communications** : ce composant permet aux différentes applications de communiquer entre elles. Différents paradigmes de communication sont fournis : communication par messages, mémoire distribuée, appel de procédures à distance...
- **Information sur les ressources** : permet d'obtenir des informations sur l'état et la structure globale du système.
- **Mécanismes de sécurité** : ce composant fournit les mécanismes d'authentification et d'autorisation des utilisateurs.

– **Création et lancement des processus** : permet de préparer l'environnement dans lequel un processus s'exécutera et de le lancer.
– **Accès aux données** : Accès performant et consistant aux données stockées dans des fichiers et des bases de données.

La table ci-dessous expose les différents services :

Service	Nom	Description
Gestion de ressources	GRAM	Allocation des ressources et gestion des processus
Communications	Nexus	Services de communication unicast et multicast
Sécurité	GSI	Authentification et autorisation
Information	MDS	Information sur la structure et l'état de la grille

TAB. 3.1 – **Principaux services offerts par Globus**

Dans [ea02] Berstis *et al* organisent les différents composants du Globus Toolkit présentés dans la table ci-dessus en trois pyramides construites sur une base commune. Cette base est le composant de sécurité (GSI) sur lequel reposent la gestion des ressources (GRAM), les mécanismes de communication (Nexus) et les services d information (MDS).

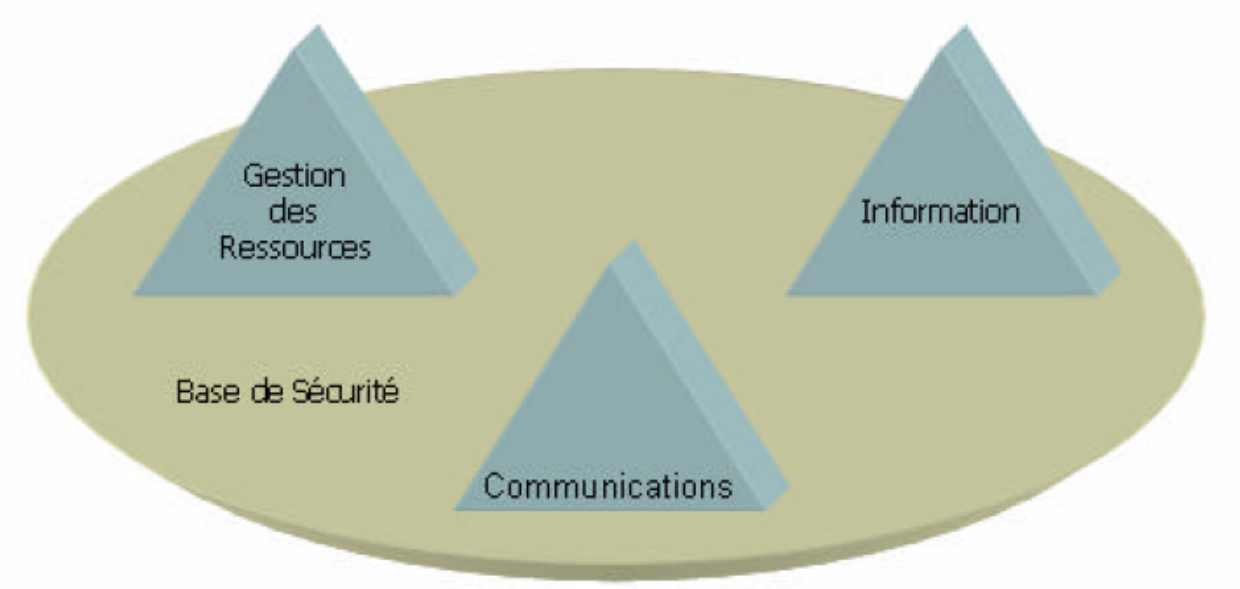

FIG. 3.1 – *Conceptualisation des services de Globus en trois pyramides (source [ea02]).*

Dans la suite de ce chapitre, nous détaillons les mécanismes de gestion des ressources, de gestion des communications et de gestion d'information et de sécurité.

3.2 Gestion des ressources

GRAM[Sou03] (« Globus Resource Allocation Manager ») est le nom du composant de Globus permettant la gestion et la supervision des ressources. Globus est basé sur une

architecture en couches avec des services de haut niveau construits à partir de services de bas niveau. Devant la multitude de services de bas niveau utilisés et la nécessité d'avoir plusieurs services de haut niveau, une architecture en sablier est adoptée[FK99].

Tout comme dans l'Internet où le protocole IP permet de masquer les différentes technologies de transmission (Ethernet, ATM, Token Ring...) aux services de haut niveau (TCP, UDP...), GRAM permet de masquer les différentes technologies de gestion de ressources de bas niveau. Ainsi les différents services de haut niveau n'ont à se préoccuper que de l'interfaçage avec GRAM.

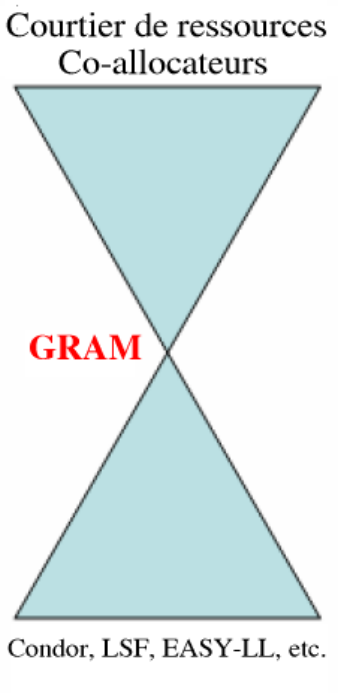

FIG. 3.2 – *Principe du sablier (source [FK99]).*

Chaque GRAM est responsable d'un ensemble de ressources opérant selon une politique commune et spécifique au site dans lequel elles se trouvent. Cette politique est souvent implémentée par un gestionnaire de ressources tel que Condor ou LSF [1]. GRAM s'interface alors avec ce système et traduit les requêtes des services de haut niveau en requêtes compréhensibles par ce gestionnaire de bas niveau.

Ainsi une grille de calcul construite avec Globus comprend une multitude de GRAM, chacun responsable d'un ensemble de ressources. De cette façon, les applications peuvent exprimer leurs besoins en ressources selon une API unifiée (celle de GRAM) et les administrateurs des ressources peuvent choisir les outils de gestion de bas niveau qui leur conviennent. Avec l'API de GRAM, les besoins en ressources sont exprimés avec un langage appelé RSL[2].

[1] **LSF** : Load Sharing Facility
[2] **RSL** : Resource Specification Language

À partir de GRAM des politiques globales de gestion de ressources (au niveau de la grille) peuvent être construites et implémentées par des courtiers (« Resource Brokers »). Un exemple d'architecture telle que définie dans[FK99] est donné dans la figure ci-après.

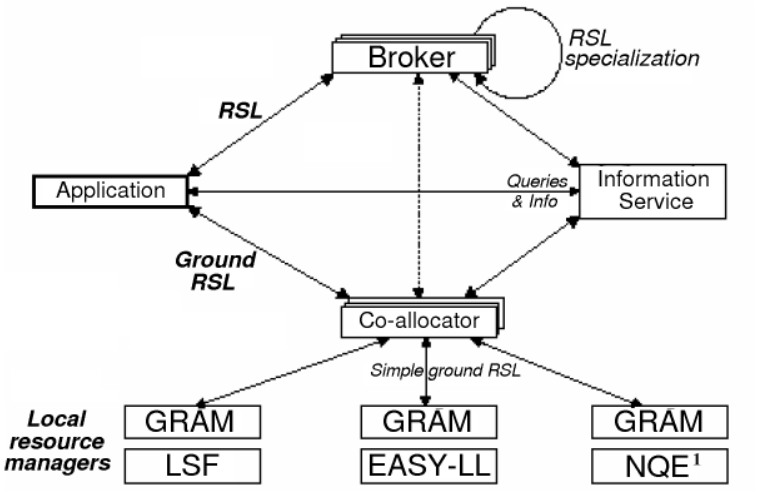

FIG. 3.3 – *Architecture de gestion de ressources de Globus (source [FK99]).*

Dans cette architecture, une application exprime ses besoins en ressources sous la forme d'une expression RSL de haut niveau. Les courtiers de ressources analysent récursivement la requête et la transforment en un ensemble de RSL bas niveau plus spécifique jusqu'à l'identification de l'ensemble des ressources susceptibles de satisfaire la demande de l'application. À ce stade de l'analyse nous avons un RSL de base (« ground RSL ») identifiant un ensemble de GRAM. Cette RSL de base est alors décomposée en RSL individuelles et chacune est envoyée au GRAM adéquat qui transforme la requête en une demande que le système sous-jacent comprend.

GRAM est donc chargé de décomposer et analyser les requêtes RSL et de lancer, superviser et terminer les tâches en cours d'exécution. Nous allons donner une vue d'ensemble rapide de RSL et pour plus de détails on pourra se référer à **www.globus.org/gram**.

RSL permet de décrire les ressources d'une façon uniforme. Il fournit les éléments syntaxiques permettant de composer des descriptions plus ou moins compliquées de ressources. La base de la syntaxe est la relation qui associe à un attribut une valeur (par exemple executable=a.out). De plus, deux constructions permettent de composer à partir des relations des descriptions plus compliquées : ce sont les requêtes groupées (« compound requests ») et les séquences de valeurs (« value sequences »). La forme la plus simple de requête groupée est la conjonction (« conjunct-request ») exprimant une conjonction au sens d'un ET logique entre relations et/ou autres requêtes groupées. Des exemples de séquences de valeurs sont la chaîne de caractères et les ensembles de valeurs.

La table 3.2 donne quelques attributs RSL utilisés par GRAM.

[1]**NQE** : Network Queuing Environment

Attribut	Description
Executable	Le nom de l'exécutable à lancer.
Arguments	Les arguments à passer au programme.
stdin, stdout, stderr	Fichiers d'entrée, de sortie et d'erreur.

TAB. 3.2 – **Exemples d'attributs RSL de GRAM**

Un exemple de requête RSL GRAM est alors :

```
(* this is a comment *)
& (executable = a.out (* <-- that is an unquoted literal *))
   (directory = /home/nobody )
   (arguments = arg1 "arg 2")
   (count = 1)
```

3.3 Gestion des communications

Les services de communications entre processus dans Globus sont assurés par la librairie de communication Nexus[Sou03]. Le principe d'architecture de Nexus est similaire à celui de GRAM : fournir une API unifiée aux services de haut niveau (MPI, RPC, CORBA...) tout en s'interfaçant avec une multitude de services de bas niveau (TCP, UDP...). Les services de communication de Nexus sont utilisés extensivement dans l'implémentation des autres services de Globus.

Les besoins des applications varient de la communication fiable par messages à la communication multipoint non fiable. IP et TCP ne sont pas en mesure de répondre à ces besoins. Nexus est alors utilisée pour combler ce manque.

Les communications dans Nexus sont définies en employant deux abstractions : les liens de communication (« Communication Links ») et les requêtes de service distant (RSR[2]).

Un lien de communication est formé par l'association d'une source (« startpoint ») et d'une destination (« endpoint »). Une opération de communication est initiée en appliquant une requête de service distant à une source. Cet appel de procédure asynchrone permet de transférer des données de la source vers toutes les destinations auxquelles elle est reliée. Plusieurs sources peuvent être associées à une destination et vice versa ce qui permet de construire des structures de communication complexes.

La figure 3.4 donne un exemple de communications Nexus entre processus.

Les liens de communication dans Nexus peuvent être transposés sur différentes méthodes de communication sous-jacentes, chacune ayant ses propres caractéristiques : protocole utilisé, sécurité, qualité de service... En associant un attribut à un lien de communication entre une source et une destination une application peut contrôler la méthode de communication employée.

[2]**RSR** : Remote Service Request

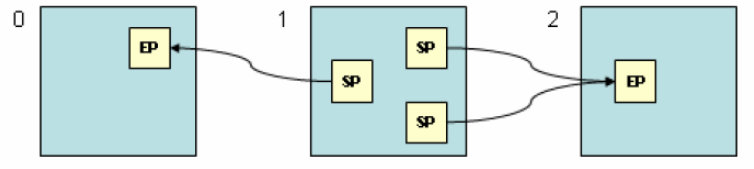

FIG. 3.4 – *Associations entre source et destination dans Nexus (source [FK99]).*

3.4 Service d'information

Les environnements de type grilles de calcul dépendent de la disponibilité d'information sur l'infrastructure sous-jacente[Sou03]. Cette information peut inclure [FK97] :

- **Configuration des ressources** : quantité de mémoire, fréquence du processeur, nombre de processeurs, nombre et type des interfaces réseau.
- **Etat instantané d'une ressource** : charge du processeur, mémoire disponible, bande passante du réseau.
- **Information sur les applications** : besoins mémoire, processeur et de stockage.

Globus offre le composant MDS[3] permettant l'accès à de telles informations. MDS offre les composants, les outils et l'API nécessaires pour la découverte, la publication et l'accès aux informations concernant la structure et l'état d'une grille de calcul. MDS contient des informations de nature statique ou dynamique. De plus, ces informations peuvent être de nature très variée ce qui permet aux développeurs de créer des applications diversifiées.

MDS utilise le standard LDAP[4] comme base pour la représentation et l'accès aux données. Le service MDS d'une grille construite avec Globus repose donc sur un ensemble de serveurs LDAP peuplant la grille.

Plus spécifiquement, MDS est constitué des modules suivants[ea02] :

« **Grid Resource Information Service** » **(GRIS)** : Les serveurs GRIS peuvent se trouver dans plusieurs endroits dans une grille. Ils peuvent contenir toute information concernant les machines qui y sont enregistrées. La nature des informations peut être statique ou dynamique. L'architecture de GRIS permet d'étendre facilement la nature des informations qu'ils peuvent contenir. Afin de ne pas charger le serveur et d'augmenter ses performances, un serveur GRIS ne contient jamais les informations concernant toutes les machines d'une grille. Ainsi pour avoir des informations sur une machine particulière l'utilisateur devra interroger le serveur GRIS adéquat. Il n'est évidemment pas efficace que l'utilisateur interroge tous les serveurs GRIS de la grille pour trouver les bonnes informations ; heureusement Globus fournit un second type de serveur pour faciliter la localisation du bon serveur GRIS.

« **Grid Index Information Service** » **(GIIS)** : Tous les serveur GRIS d'une grille sont enregistrés lors de leur démarrage avec un serveur GIIS. Ce serveur contient des informations concernant la localisation dans la grille des serveurs GRIS et les noms

[3]**MDS** : Metacomputing Directory Service
[4]**LDAP** : Lightweight Directory Access Protocol

des machines enregistrées avec chaque GRIS. Ainsi un utilisateur peut avoir des informations concernant une machine particulière en contactant le serveur GIIS. Un serveur GIIS peut aussi contenir (dans un cache) certaines informations contenues dans les serveurs GRIS. Un seul serveur GIIS dans une grille constitue un point fragile. Pour cela des serveurs secondaires sont mis en place pour assurer une bonne tolérance aux pannes. D'autre part les serveurs GIIS peuvent être organisés selon une hiérarchie comme le système DNS.

Fournisseur d'information (« Information Provider ») : Ce composant assure la traduction des propriétés et du status des ressources selon le schéma et les fichiers de configuration de MDS (« MDS schema »).

Client MDS : ce composant permet d'interroger MDS pour obtenir des informations concernant une ressource.

La figure 3.5 illustre le modèle conceptuel de MDS.

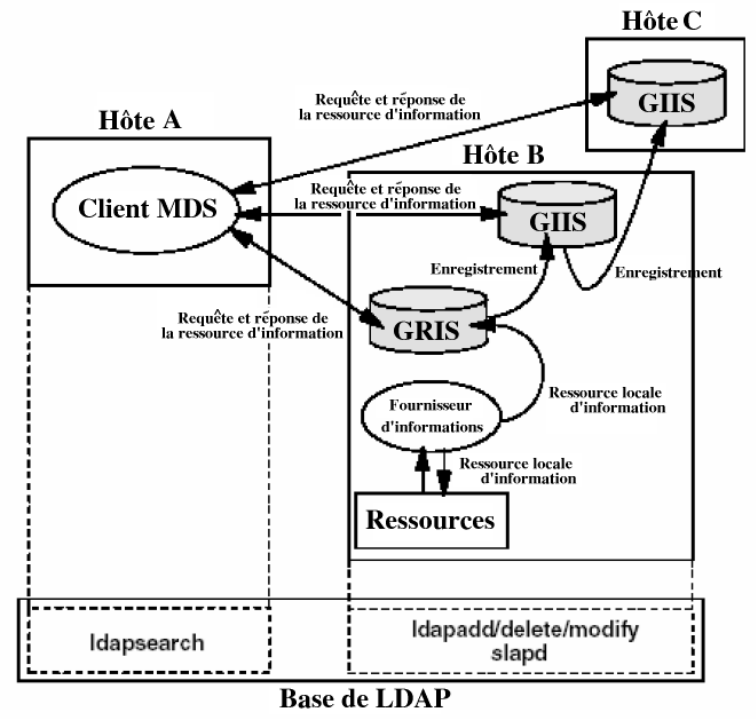

FIG. 3.5 – *Modèle conceptuel de MDS (source [ea02]).*

MDS modélise les caractéristiques physiques et logiques d'une ressource par une hiérarchie d'éléments. Cette hiérarchie est composée d'un ensemble de classes d'objets et d'attributs (conformément aux principes de LDAP). Des exemples d'objets et d'attributs sont donnés dans la table 3.3.

Object class MdsHost : représente un noeud (calculateur) du réseau.
Attribute type Mds-Host-hn : indique le nom du noeud.
Object class MdsDevice : représente un élément (device) du noeud.
Attribute type Mds-Device-name : indique le nom de l'élément.
Object class MdsOs : représente le système d'exploitation installé sur une machine.
Attribute type Mds-Os-name : indique le nom du système d'exploitation.
Attribute type Mds-Os-release : indique la version du SE.
Attribute type Mds-Os-version : indique la version du noyau (Kernel) du SE.
Object class MdsCpu : représente un processeur (CPU).
Attribute type Mds-Cpu-vendor : indique le nom du constructeur du processeur.
Attribute type Mds-Cpu-model : indique le modèle du CPU.
Attribute type Mds-Cpu-version : indique la version du CPU.
Attribute type Mds-Cpu-features : indique les fonctionnalités du CPU.
Attribute type Mds-Cpu-speedMHz : indique la fréquence du CPU.

TAB. 3.3 – **Quelques éléments de la hiérarchie d'information MDS.**

3.5 Services de sécurité

Globus propose une architecture de sécurité [Sou03] complexe permettant de sécuriser le fonctionnement de la grille. Les composants de sécurité fournissent les mécanismes qui assurent l'authentification, l'autorisation et la confidentialité des échanges. Cette architecture de sécurité est appelée GSI[5].

Dans la suite nous supposons que le lecteur est familier avec les concepts de base de la sécurité et de la cryptographie : cryptographie symétrique (à clé secrète), cryptographie asymétrique (à clé publique), autorité de certification, certificats numériques...

Globus repose sur la cryptographie à clé publique. Ainsi pour pouvoir utiliser les mécanismes de sécurité de GSI, il faut créer une paire de clés (publique/privée) et demander la délivrance d'un certificat numérique d'une autorité de certification (AC) et cela pour chaque noeud de la grille. À la fin de cette procédure, nous avons sur chaque machine trois fichiers importants contenant respectivement : la clé publique de l'AC, la clé privée du noeud et le certificat numérique du noeud. Il est important de noter que le fichier contenant la clé privée du noeud devra être particulièrement sécurisé pour ne pas y permettre un accès illicite par des utilisateurs non autorisés. Pour cela, les mécanismes du système d'exploitation sous-jacent devront être utilisés (droits d'accès, chiffrement, détection d'intrusion et d'intégrité). De plus, la clé privée n'est fonctionnelle qu'avec l'introduction par l'utilisateur d'un mot de passe qui permet de la compléter ; cela permet d'introduire une couche de sécurité supplémentaire.

[5]**GSI** : Grid Security Infrastructure

3.5.1 Authentification et autorisation

Dans un scénario de communication entre deux noeuds de la grille, il faut qu'ils aient confiance l'un en l'autre[Sou03]. Pour cela GSI offre un mécanisme d'authentification mutuelle (« mutual authentication ») et d'autorisation qui utilise SSL/TLS comme base. Globus assure cette procédure pour chaque requête d'exécution de tâche. La procédure est la suivante[ea02] :

1. Le noeud A envoie son certificat au noeud B.
2. B s'assure que le certificat est valide et extrait l'identité et la clé publique de A du certificat.
3. B génère un nombre aléatoire et l'envoie à A.
4. Lors de la réception de ce nombre, A le chiffre avec sa clé privée (c'est ici que A pourra demander à l'utilisateur d'entrer le mot de passe) et l'envoie à B.
5. Lors de la réception du nombre chiffré, B le déchiffre avec la clé publique de A et s'assure qu'il est le même. A est alors authentifié par B.
6. La procédure est répétée dans le sens inverse pour que A authentifie B.
7. B transpose le nom de l'utilisateur se trouvant dans le certificat en un nom d'utilisateur local au noeud. Pour cela un fichier appelé grid-map est utilisé. Il contient une entrée ressemblant à **"/O=Grid/O=Globus/OU=enst.fr/CN=Un Tel" untel**. Cette ligne permet de transposer le DN[6] de Monsieur Un Tel en un nom d'utilisateur local (untel). Cette dernière étape constitue la phase d'autorisation.

3.5.2 Délégation et Single Sign-On

Imaginons la situation où un utilisateur devra lancer des milliers de tâches et que ses tâches devront lancer à leur tour de nouvelles tâches. Il devient impossible de demander à l'utilisateur de fournir son mot de passe. Heureusement GSI offre un mécanisme de délégation permettant de diminuer le nombre de fois qu'un utilisateur devra fournir son mot de passe. Ce mécanisme de délégation est une extension des mécanismes de SSL.

La création d'un proxy est la solution. Ainsi si un utilisateur utilise un noeud A authentifié par un noeud B, il pourra créer un proxy sur B lui délégant ainsi son autorité. B sera en mesure de créer des tâches sur un noeud C comme si l'utilisateur sur A les avait créé. Un proxy consiste en un nouveau certificat numérique avec une nouvelle paire de clés publique/privée. Il contient l'identité de l'utilisateur légèrement modifiée pour indiquer que c'est un proxy. Ce nouveau certificat est signé par l'utilisateur lui-même et non pas par l'AC. De plus un proxy a une durée de vie limitée. Il est alors utilisé pour assurer la procédure d'authentification mutuelle et d'autorisation sans avoir à impliquer l'utilisateur.

Il faut noter que ce processus d'authentification et d'autorisation du proxy nécessite l'établissement d'une chaîne de confiance (chain of trust) de l'autorité de certification jusqu'au proxy en passant par l'utilisateur.

[6] **DN** : Distinguished Name

3.5.3 Communication chiffrée

Bien que GSI requiert le déroulement des processus d'authentification et d'autorisation, la procédure de chiffrement des communications entre noeuds n'est pas utilisée par défaut et cela afin d'éviter la surcharge des échanges par le chiffrement et le déchiffrement. Mais puisque GSI utilise SSL comme système sous-jacent, la procédure de création d'une clé secrète commune entre deux noeuds est possible. Cette clé sera alors utilisée par un algorithme tel que DES pour chiffrer les échanges. D'un autre côté GSI assure par défaut l'intégrité des données.

3.6 Conclusion

Dans ce chapitre nous avons présenté Globus[Sou03], qui constitue une boîte à outils permettant la construction de grilles. À part les services et les composants de base que nous avons détaillé ci-dessus, Globus offre une multitude de services et d'outils de haut niveau. Nous allons décrire brièvement quelques uns, notamment GridFTP, GASS et HBM.

GridFTP permet de transférer des données entre les noeuds d'une grille d'une façon fiable et sécurisée. GridFTP étend FTP avec des fonctionnalités adaptées aux grilles telles que les transferts multi-flux, partiels, redémarrables ainsi que la possibilité de négocier les paramètres des connexions TCP (taille de la fenêtre de congestion et des tampons de réception) et la sécurité des transferts. GridFTP est encore en cours de normalisation (draft), ainsi Globus ne supporte pas toutes ses fonctionnalités actuelles.

Le « Heart Beat Monitor » ou HBM est conçu pour fournir un système de supervision des processus à distance et la détection des pannes (remote fault monitoring). Il consiste en un simple système de scrutation (« polling ») : un processus s'enregistre avec le service qui s'attend à recevoir régulièrement des messages appelés battements de coeur (« heartbeats ») de la part du processus. Si un battement n'est pas reçu, HBM tente de déterminer si le processus est en arrêt ou si le réseau sous-jacent l'est.

Le « Global Access to Secondary Storage » ou GASS est un service permettant l'accès à distance aux fichiers (Remote file I/O). Ce composant permet aux processus de lire et écrire des fichiers distants sans avoir à changer leurs appels systèmes traditionnels. GASS repose sur l'utilisation des mémoires caches locales lors de la lecture et l'écriture dans les fichiers afin d'augmenter les performances. Le système détermine quand un fichier n'est plus référencé pour mettre à jour la copie distante.

En plus de ces services, Globus offre plusieurs outils permettant aux développeurs d'applications d'écrire des programmes pour la grille. Comme exemple d'outils nous pouvons citer MPICH une version de MPI pour la grille, Nimrod-G permettant de traiter efficacement les problèmes de larges ensembles de paramètres...

Enfin nous notons que Globus est toujours un projet en constante évolution et la communauté, qu'elle soit académique ou industrielle, y participe activement. On y trouve

des entreprises telles que IBM et Platform Computing qui tentent de créer des produits commerciaux basés sur Globus plus complets et faciles à installer et à administrer.

Chapitre 4

Portage d'une application MPI sous Globus

Dans ce chapitre, nous présentons le portage d'une application MPI sous Globus, application ayant déjà fait ses preuves sous LAM/MPICH.

Tout d'abord, nous décrivons brièvement le programme MPI choisi, puis nous déroulons les deux algorithmes associés à cette application (synchrone et asynchrone) et enfin, nous présentons les différents résultats obtenus lors des différentes expérimentations.

4.1 Présentation de l'application : le problème cinétique d'advection-diffusion avec deux espèces chimiques

4.1.1 Principes

Le problème correspond à un grand système d'équations différentielles partielles (PDEs[1]) [JBV04]. Il modélise un mécanisme de cinétique chimique d'advection-diffusion.

On cherche à déterminer les évolutions en fonction du temps, des concentrations de deux espèces chimiques en tout point de l'espace à deux dimensions considéré. On effectue donc une discrétisation de cet espace afin de créer une *grille*.

Cette discrétisation de l'espace considéré permet donc de transformer le système de PDEs pour obtenir un système d'ODEs[2] de type :

$$\frac{dy(t)}{dt} = f(y(t),\ t) \tag{4.1}$$

4.1.2 Déroulement de l'algorithme séquentiel

La grille de discrétisation des deux espèces chimiques obtenue est représentée par un vecteur y [JBV04].

[1] **PDEs** : Partial Differential Equations

[2] **ODEs** : Ordinary Differential Equations

Une fois ce vecteur construit, la démarche pour résoudre le système d'ODEs consiste à exécuter l'algorithme en plusieurs pas de temps, et à découper chacun de ces pas de temps en trois étapes :

- discrétisation de l'ODE avec la méthode implicite d'Euler,
- déduction d'un système linéaire avec la méthode de Newton,
- et résolution de ce système avec la méthode $GMRES$ [3].

Les constantes du problème (pas de temps et intervalle de résolution) sont donnés. On fixe également un ε premettant de détecter la convergence du système linéaire déduit de la méthode de Newton.

4.1.3 Préliminaires sur les algorithmes parallèles

Pour l'implantation de ces deux algorithmes, on considère que la machine virtuelle est constituée de α processeurs numérotés de 0 à $\alpha - 1$. Chaque processeur peut envoyer et recevoir des données de tous les autres.

Nous considérons dans la suite de ce chapitre qu'une seule tâche s'exécute sur chaque processeur. Nous utiliserons ainsi les notions de « tâches », « nœuds », « processeurs », « sites » ou encore « hôtes » sans distinction particulière.

Répartition des données et dépendances entre sites

La répartition des données choisie ici est celle proposée dans [JBV04]. Il est montré qu'avec une telle répartition le calcul d'une composante de la grille au pas de temps suivant dépend toujours des mêmes éléments (les éléments courants, ainsi que ceux placés au Sud, au Nord, à l'Est ou à l'Ouest). Ceci nous permet de déduire que le calcul d'une composante de la grille au pas de temps suivant nécessite au plus deux valeurs calculées par un autre processeur. Ces éventuelles valeurs sont les concentrations de c^1 et de c^2 (les deux espèces chimiques) des sites voisins situées sur la rangée contiguë aux valeurs locales.

Ces algorithmes parallèles sont tout particulièrement adaptés aux calculs sur grille, les résultats du DEA de Philippe Vuillemin [Vui03] le prouvent. Nous avons alors utilisé Globus pour que ces algorithmes s'exécutent sur une grille gérer à l'aide de ce middleware.

4.2 Déroulement de l'algorithme synchrone

Phase 1 : Initialisation des itérations

On fixe pour chacun des processeurs les conditions initiales des itérations comme le précise la formule (4.2). Pour cela, on assigne à chaque sous-vecteur de y^0 les valeurs des composantes calculées localement au pas de temps précédent ($y(0)$ local si on est au premier pas de temps).

[3] **GMRES** : Generalized Minimum RESidual

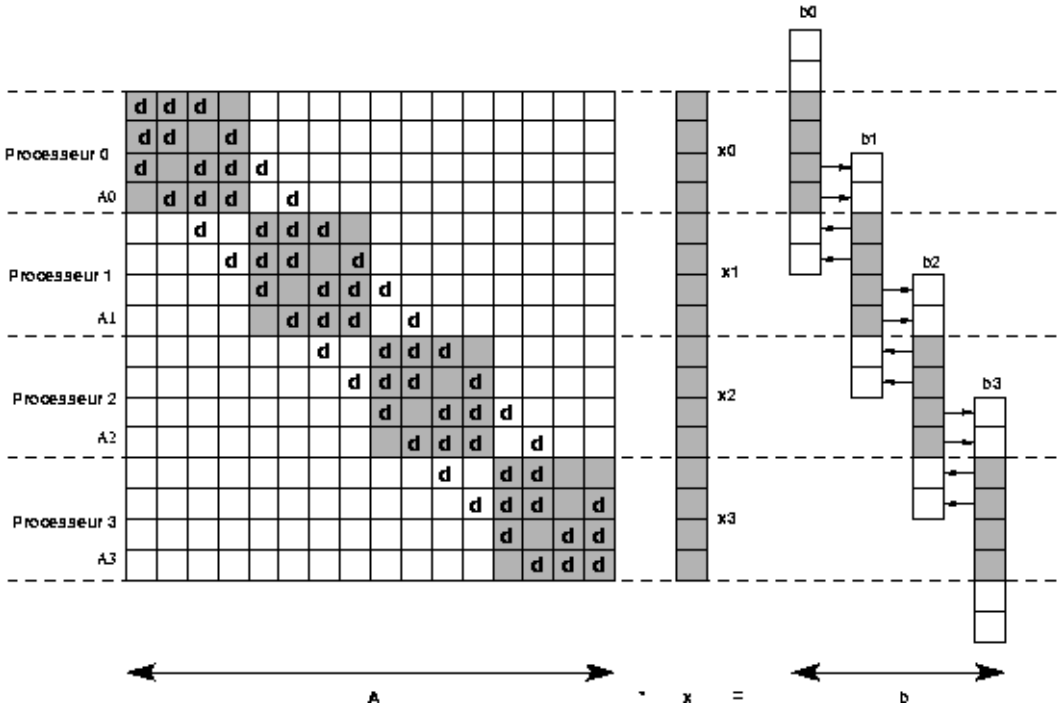

FIG. 4.1 – *Découpage des données pour la résolution du système linéaire sur plusieurs processeurs.*

$$y(t+h)^0 = y(t) \tag{4.2}$$

Phase 2 : Calcul de la matrice A

À la première itération effectuée, chaque processeur déduit de ses composantes locales de y une partie de la matrice A. Pour la sous-matrice considérée, l'exploitation de certains éléments non nuls nécessite uniquement des valeurs de y locales (celles en gris dans la figure 4.1). D'autres en revanche font intervenir des composantes calculées par d'autres sites.

Phase 3 : Calcul du vecteur b

À chaque nouvelle itération, on calcule la partie locale du vecteur b avec F selon la formule (4.3). Celui-ci est donc déduit à partir de :

- la valeur de y calculée localement à l'itération précédente,
- et aussi des composantes locales de y obtenues au pas de temps précédent ($y(0)$ local si on est au premier pas de temps).

$$F(y(t+h),\ y(t),\ t) = y(t) + h * f(y(t+h),\ t) - y(t+h) \tag{4.3}$$

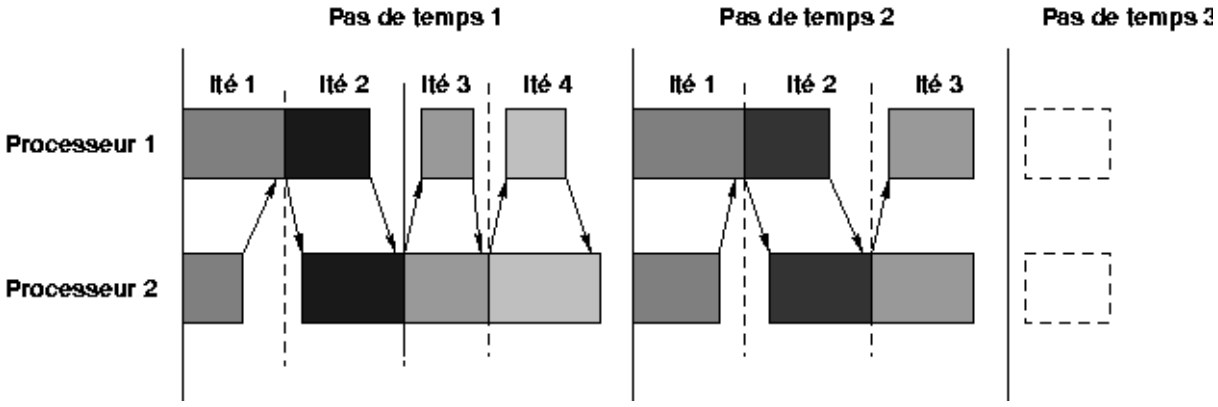

FIG. 4.2 – *Exécution synchrone.*

Phase 4 : Résolution du système linéaire et diffusion des dépendances

Une fois que chaque site a calculé sa sous-matrice de A et ses sous-vecteurs de b, tous les processeurs se synchronisent pour lancer en même temps une itération. La méthode $GMRES$ est donc exécutée sur chaque site avec les valeurs disponibles localement. Cette méthode permet de trouver les n solutions x_i d'un système linéaire de taille $n \times n$, c'est une des méthodes les plus rapides pour la résolution itérative d'un système linéaire.

Chaque tâche est responsable de l'approximation des composantes locales de l'inconnue x du système. Ensuite les processeurs s'échangent les valeurs effectives des dépendances. Ici, les réceptions sont effectuées de manière *bloquante* afin de mettre en valeur le caractère synchrone des itérations.

Phase 5 : Détection de convergence et procédure d'arrêt

Si la convergence globale du système est détectée, chaque tâche peut retourner son approximation des composantes locales de y. Il y a ensuite synchronisation entre les processeurs pour effectuer une nouvelle méthode implicite d'Euler afin de calculer le vecteur y localement au pas suivant.

Si la convergence n'est pas encore détectée, les processeurs se synchronisent et reprennent l'algorithme à la phase 3.

Lorsque l'on arrive à la fin de l'intervalle de temps T, les vecteurs locaux de tous les sites ont été calculés pour chaque pas de temps, on peut donc interrompre l'activité de toutes les tâches.

Ceci est décrit dans la figure 4.2 pour les deux premiers pas de temps d'une exécution. On remarque que les pas de temps sont tous synchronisés et il en est de même pour les itérations s'effectuant au cours d'un pas de temps. On remarque également, entre les itérations, des périodes d'inactivité en terme de calcul représentées par les espaces blancs et qui retardent la progression des processeurs.

4.3 Déroulement de l'algorithme asynchrone

L'implantation de l'algorithme dans sa version asynchrone est très similaire. Pour commencer, le calcul local de $y(0)$ est effectué de la même manière qu'en synchrone.

Ensuite les processeurs se synchronisent également à chaque pas de temps tout au long de l'intervalle de temps T pour effectuer les résolutions d'ODEs. Puis chaque pas de temps se déroule de la manière suivante :

Les phases concernant l'initialisation des itérations, le calcul de la matrice A et du vecteur b (les phases 1 à 3) se déroulent exactement de la même manière que dans le cas synchrone. Leur description est donc exposée dans le paragraphe 4.2.

Phase 4 : Résolution du système linéaire et diffusion des dépendances

Dans la version asynchrone, seule la première itération démarre de manière synchrone sur tous les sites. Par la suite, il n'y a aucune synchronisation entre les itérations et le calcul des composantes locales de x s'effectue à partir des dernières valeurs disponibles localement, aussi anciennes soient-elles.

Après une itération, les dépendances sont envoyées comme pour la version synchrone, mais les réceptions sont non bloquantes. Un site peut donc commencer si nécessaire l'itération suivante sans attendre d'avoir reçu les données sollicitées.

Phase 5 : Détection de convergence et procédure d'arrêt

Si la convergence globale du système est détectée, chaque tâche peut retourner son approximation des composantes locales de y. Il y a ensuite synchronisation entre les processeurs pour effectuer une nouvelle méthode implicite d'Euler afin de calculer le vecteur y localement au pas suivant.

Sinon, il n'y a pas convergence et les processeurs se synchronisent et reprennent l'algorithme à la phase 3.

Lorsque l'on arrive à la fin de l'intervalle de temps T, les vecteurs locaux de tous les sites ont été calculés pour chaque pas de temps. On peut donc interrompre l'activité de toutes les tâches.

Ceci est décrit dans la figure 4.3 pour les deux premiers pas de temps d'une exécution. On remarque que les pas de temps sont tous synchronisés mais que les itérations s'effectuent sans aucune synchronisation (à part pour la première de chaque pas de temps). On remarque également que l'exécution d'un pas de temps s'effectue en moins de temps que pour le cas synchrone (car il y a peu de périodes d'inactivité en terme de calcul, même si plus d'itérations sont effectuées dans cette version que dans la précédente).

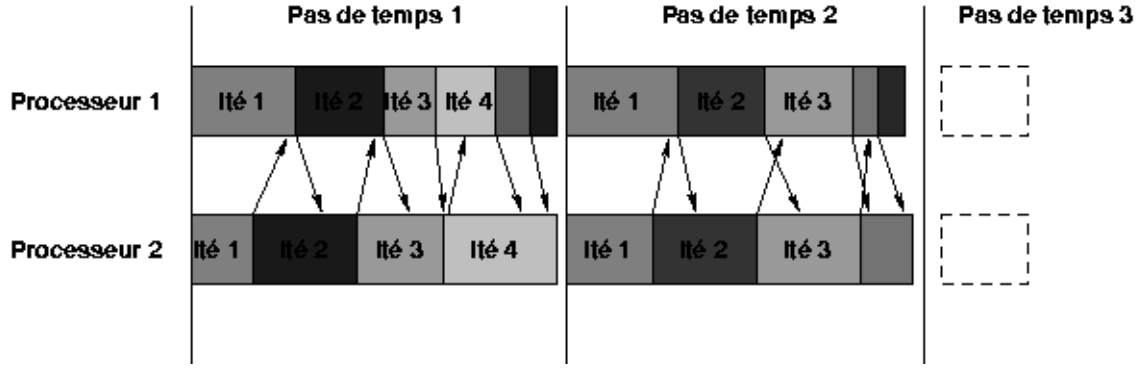

FIG. 4.3 – *Exécution asynchrone.*

4.4 Portage de l'application

4.4.1 Existant

L'application synchrone utilise l'implémentation LAM de MPI. Cette implémentation ne gérant pas l'asynchronisme, la version asynchrone de l'application a été implanté à l'aide de MPICH. Ce dernier propose des modifications de MPI au niveau de la gestion des threads (**MARCEL** [AM03, AMN01]) et de la gestion des communications (**MADELEINE** [AMN01]).

4.4.2 Gestion des communications

Que ce soit pour la version synchrone ou asynchrone, l'ensemble des communications de l'application est géré au niveau de l'implémentation MPI utilisée.

4.4.3 Services de sécurité

Globus gère l'ensemble des processus d'authentification utilisateur. Ainsi, pour qu'un utilisateur puisse exécuter l'application, il doit avoir, au préalable, validé un certificat d'authentification auprès du gestionnaire de la grille (cf paragraphe 3.5.1).

4.4.4 Gestion des ressources

Pour que les versions synchrones et asynchrones de l'application puissent être interprétées par le middleware Globus, il a fallu écrire deux fichier RSL (cf paragraphe 3.2).

En ce qui concerne la version synchrone, il s'agissait juste de spécifier que l'application était écrite en MPI à l'aide de l'option **jobType** et de compléter les diverses options nécessaires (cf listing 4.1).

Listing 4.1 – *advecdiffu_sync.rsl*

```
& (count=9)
  (jobType=mpi)
  (project="MPI advecdiffu_sync3")
```

```
  (environment=
    (DEFPART IAPAR)
    (GLOBUS_DUROC_SUBJOB_INDEX 0)
    (LD_LIBRARY_PATH /home1/globus/gt3/lib/)
    (GLOBUS_TCP_PORT_RANGE "3000,3200")
  )
  (directory=/home/clement/grid/test_mpi)
  (executable=/home/clement/grid/test_mpi/advecdiffu_sync3)
  (stderr=/home/clement/grid/test_mpi/resultat_glo.err)
  (stdout=/home/clement/grid/test_mpi/resultat_glo.out)
```

Les arguments passés à la commande **leonie** lui spécifient juste l'application à exécuter, la flavor a utilisée et la liste des machine disponibles (cf listing 4.2).

Listing 4.2 – *advecdiffu_async.rsl*

```
& (count=9)
  (jobType=single)
  (project="MPI advecdiffu_sync3")
  (environment=
    (DEFPART IAPAR)
    (GLOBUS_DUROC_SUBJOB_INDEX 0)
    (LD_LIBRARY_PATH /home1/globus/gt3/lib/)
    (PM2_HOME "/home/clement/grid/test_pm2")
    (PM2_ROOT "/opt/grid/pm2")
    (PM2_BUILD_DIR "/home/clement/grid/test_pm2/build")
    (PM2_FLAVOR pm3)
    (MARCEL_ROOT "/opt/grid/pm2/marcel")
    (MADELEINE_ROOT "/opt/grid/pm2/madeleine")
    (PATH "/opt/grid/gcc-3.3.2/bin:/opt/grid/lam-7.0.3/bin
      :/opt/grid/Python-2.3/bin:/home1/globus/gt3/bin
      :/home1/globus/gt3/sbin:/usr/local/bin:/usr/bin:/bin
      :/usr/bin/X11:/usr/games:/usr/local1/bin
      :/usr/local1/j2sdk1.4.2_03/bin:/usr/local1/apache-ant-1.6.1/bin
      :/opt/grid/pm2/bin:/opt/grid/pm2/marcel/bin
      :/opt/grid/pm2/madeleine/bin")
    (PM2_RSH ssh)
    (PM2_SSH ssh)
    (LEO_SSH ssh)
    (LEO_RSH ssh)
  )
  (directory=/home/clement/grid/test_mpi)
  (executable=/opt/grid/pm2/bin/leonie)
  (arguments=
    "--x"
    "--appli=/home/clement/grid/test_mpi/advecdiffu_async3_mpi"
    "--flavor=mpi-flav"
    "--net=/home/clement/grid/test_mpi/networks.cfg"
    "/home/clement/grid/test_mpi/mpi-conf-small.cfg")
  (stderr=/home/clement/grid/test_mpi/asyncGlo.err)
```

```
(stdout=/home/clement/grid/test_mpi/asyncGlo.out)
```

Ressource Broker

Par défaut, l'affectation des différents processus aux différentes machines est réalisé de manière quasi statique par MPI ou Globus (GRAM). Cependant, afin d'obtenir un niveau de transparence plus élevé, il est souhaitable de disposer d'un service de gestion de ressources dynamique tel que Nimrod/G [BAG00]. Dans notre cas, Nimrod/G n'a pu nous satisfaire car la requête qu'il utilise se fait au niveau *local* (GRIS) et non au niveau souhaité. Dans notre exemple, il effectuait la requête au niveau du GRIS « grappa » et par conséquent ne trouvait que la machine *grappa* (cf fig 4.4). Le fait de ne pas pouvoir, pour le moment, paramétrer cette requête nous a obligé a mettre de côté cette application.

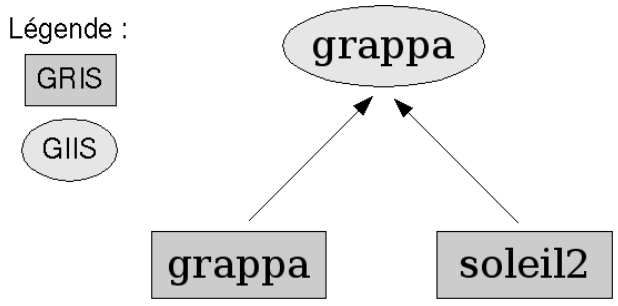

FIG. 4.4 – *Schéma de l'architecture virtuelle de la plateforme de test.*

4.5 Expérimentations

4.5.1 Contexte général des expérimentations

Dans l'expérimentation qui suit, nous comparons les résultats d'exécution des versions synchrone et asynchrone de l'application d'advection-diffusion dans les environnements LAM/MPICH et Globus. Les temps reportés dans les tableaux représentent les moyennes des résultats obtenus suite aux différentes séries de tests.

Ces tests ont été réalisés sous Linux Debian. La machine parallèle virtuelle employée est composée de 9 postes de travail connectés entre eux par l'intermédiaire d'un réseau Ethernet 100Mb/s.

4.5.2 Résultats des expérimentations

Nous avons étudié le comportement des applications synchrones et asynchrones dans les environnements Globus et LAM/MPICH en fonction de la taille de la matrice (de N=360 à N=720, avec N la taille de la matrice).

Les résultats des expérimentations sont donnés dans le tableau 4.1, où les temps d'exécution sont en secondes.

Taille de la matrice	**Temps LAM/MPICH (en s)**		**Temps Globus (en s)**	
	Synchrone	Asynchrone	Synchrone	Asynchrone
360	127	59	133 (+6)	67 (+8)
450	158	105	166 (+8)	120 (+15)
540	223	168	236 (+13)	195 (+27)
630	293	245	297 (+4)	260 (+15)
720	406	322	413 (+7)	330 (+8)

TAB. 4.1 – *Résultats de l'expérimentation, comparant Globus et LAM/MPICH.*

Nous pouvons remarquer dans les figures 4.5 et 4.6 que les performances du middleware Globus sont voisines de LAM/MPICH, mais toujours inférieures. Pour exemple, prenons les résultats obtenus pour une taille de matrice de 720. Les résultats de Globus sont supérieurs de $1,72\%$ et $2,48\%$ respectivement pour les versions synchrone et asynchrone, ce qui paraît négligeable pour des calculs dont la durée oscille entre 5 et 7 minutes.

De plus, force est de constater que la version asynchrone est nettement plus efficace que la version synchrone (figure 4.7). Pour une taille de la matrice équivalente à 360, on atteind des valeurs qui sont 2 fois moins grandes pour la version asynchrone que pour la version synchrone.

Des expérimentations sont actuellement en cours sur un plus grand nombre de machines (les premiers tests montrent qu'avec 25 machines le temps de l'algorithme synchrone sous LAM, pour une matrice de taille 720, passe de 406 secondes à 160).

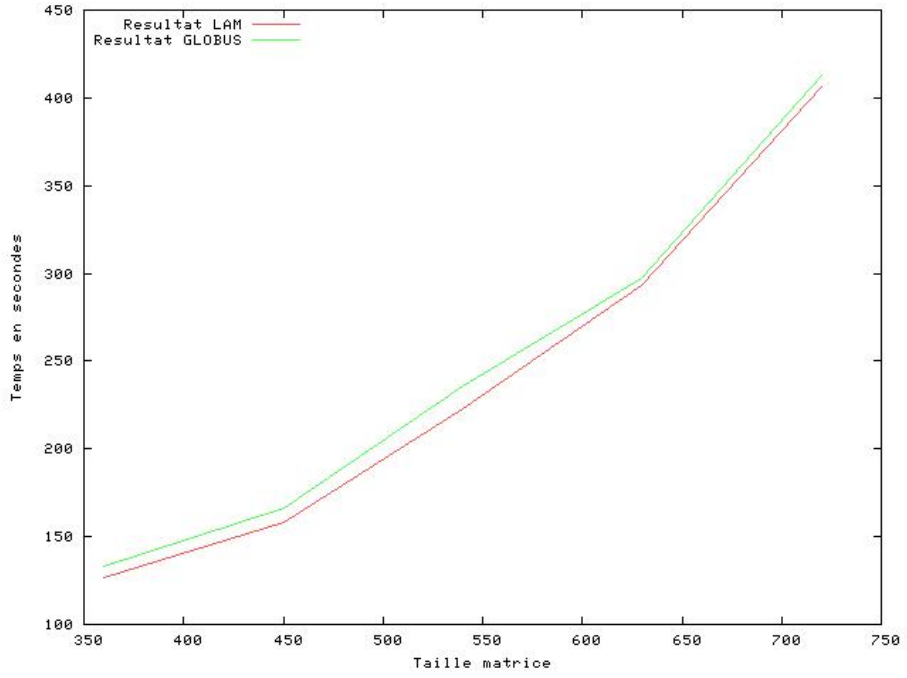

FIG. 4.5 – *Performances des middlewares LAM et Globus sur l'application d'advection-diffusion version synchrone.*

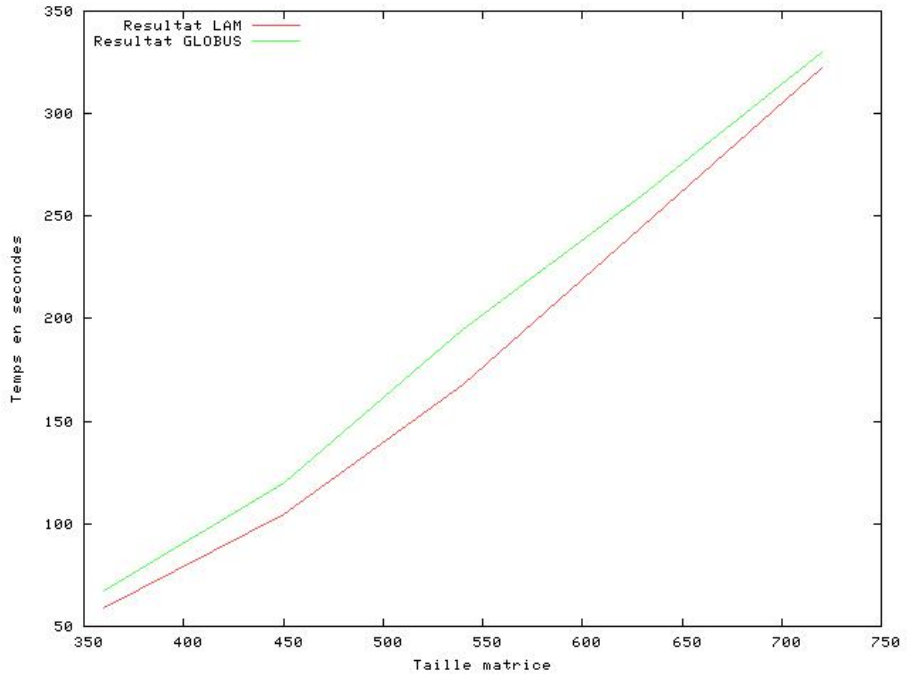

FIG. 4.6 – *Performances des middlewares MPICH et Globus sur l'application d'advection-diffusion version asynchrone.*

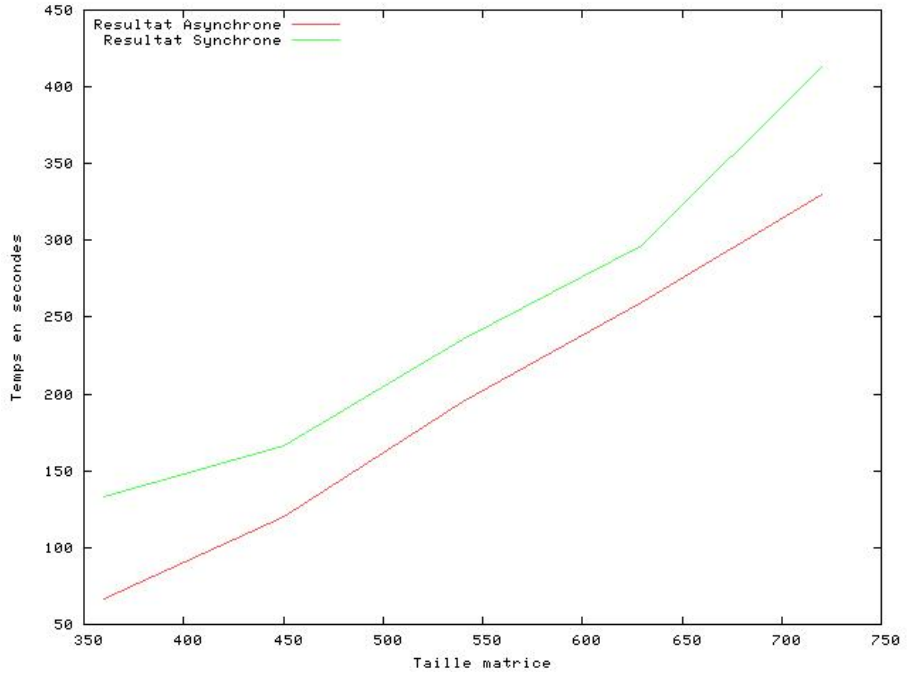

FIG. 4.7 – *Performances des algorithmes synchrones et asynchrones avec le middleware Globus.*

4.6 Conclusions sur le portage de l'application numérique asynchrone MPI sous le middleware Globus

Tout d'abord, nous avons commencé par installer Globus sur une machine. Cette installation fut plutôt longue et fastidieuse, cela est surtout dû au fait que les documentations associées à la version 3.0 du middleware n'étaient pas assez claires et que Globus, malgré une ancienneté relative (début du projet en 1997), est un produit jeune qui ne cesse d'évoluer. Pour preuve, durant mon stage une nouvelle version du produit (version 3.2) a vu le jour et les développeurs projettent de créer trois nouvelles versions d'ici la fin de l'année pour arriver à la version 4.2.

Toutefois, une de leur priorité, pour cette année 2004, était de compléter, ordonner et clarifier leurs documentations, ce qui semble être en parti fait avec la documentation associée à leur dernière version.

Malgré les différents problèmes survenus et précités, le portage de l'application MPI se fit de manière simple et rapide, ce qui nous a permis d'obtenir des résultats concrets et satisfaisants.

Néanmoins, il reste très difficile d'appréhender exactement ce que prennent en charge les différents composants de GLOBUS (en particulier au niveau des communications et de la gestion des ressources) par rapport à MPI.

Les expérimentations, dans le cas du problème cinétique d'advection-diffusion, permettent d'émettre certaines conclusions sur l'efficacité de Globus par rapport à LAM/MPICH et sur la différence entre les versions asynchrone et synchrone du problème.

Une des conclusions qui s'impose à nous est que le middleware Globus est tout à fait compétitif par rapport à LAM/MPICH avec des temps d'exécution très satisfaisants. La différence par rapport à LAM/MPICH et surtout due au fait que Globus est très sécurisé et qu'il faut un certain temps au middleware pour authentifier et autoriser un utilisateur à exécuter une application sur la grille.

Ces expérimentations, avec les deux environnements, permettent de montrer avec netteté que les performances de l'algorithme asynchrone sont supérieures à celles de l'algorithme synchrone. Ces résultats rejoignent ceux trouvés lors du stage de DEA de Philippe Vuillemin [Vui03] et renforcent le fait que les algorithmes asynchrones sont tout particulièrement adaptés aux calculs sur grille.

Conclusion

Au cours de ce stage de DEA, nous avons pu tester les capacités du middleware Globus à l'aide de l'application d'advection-diffusion. Les conclusions que nous en avons retiré sont telles qu'en termes de résultats Globus est très proche de LAM.

Nous avons vu que Globus était un middleware jeune est voué à une évolution rapide, notamment au cours de cette année 2004. Il faudra vérifier que son évolution va dans notre sens et qu'elle nous permettra d'utiliser pleinement les futurs versions de Globus.

Le paragraphe suivant donne un aperçu des points qu'il reste à étudier.

Nous avons mis en évidence que les deux middleware, Globus et LAM, ont des résultats proches l'un de l'autre. Pour pouvoir pencher en la faveur de l'un ou de l'autre de nombreuses choses restent à tester.

Tous les tests que nous avons pu faire ici, l'ont été sur des machines proches physiquement et avec des charges réseau et des charges machines quasi nulles. Il faudrait, pour tester la robustesse de Globus, effectuer de nouveaux tests, avec des machines provenant de plusieurs sites distants et avec des conditions d'exécution différentes (charge artificielle des machines et/ou de la ligne Ethernet).

En ce qui concerne le gestionnaire de ressources, il faudrait, une fois qu'une nouvelle version de Nimrod/G sera opérationnelle, pouvoir la tester afin de constater l'utilité ou non d'un tel logiciel.

Annexe A

Installation du Globus Toolkit 3.0

La première partie de mon travail a été d'installer sur une machine le Globus Toolkit[Hed03] [DS04][Sot03][Bac03][LF03]. J'ai choisi la dernière version stable du produit, soit la version 3.0, téléchargeable sur le site officiel de globus.

A.1 Les prérequis

Avant toute chose, l'installation du Globus Toolkit necéssite certains prérequis, qu'il a fallu s'assurer d'installer pour le bon fonctionnement du produit.

A.1.1 La machine virtuelle JAVA

L'installation d'une version de java est primordiale. Il ne faut pas oublier d'exporter le chemin d'installation dans la variable **JAVA_HOME** et de mettre à jour la variable **PATH**.

Ici, j'ai installé la version 1.4.2_03 de la j2sdk. Auparavant, j'avais essayé une version trop récente et encore en version beta, ce qui a eu pour effet un échec lors de l'installation de Globus.

Une fois la première étape effectuée, il faut créer un lien nommé **java** dans le répertoire d'installation de Globus.

Les différentes version de java peuvent être télecharger sur le site de sun à l'adresse suivante : http ://www.sun.com

A.1.2 Junit

Junit est le framework Java open source pour la réalisation de tests unitaires. Pour une explication rapide, on pourra se référer à la page de Jérôme Cheynet ingénieur en Informatique et Réseaux.

Installation de **Junit** :

1. télécharger l'archive **junit3.8.1.zip** ;
2. la décompresser ;
3. et mettre à jour la variable CLASSPATH comme suit :

```
export CLASSPATH=$CLASSPATH:$GLOBUS_LOCATION/junit/junit.jar
```

A.1.3 Tomcat

Installation de **Tomcat** :

1. télécharger **tomcat** ;
2. décompresser l'archive ;
3. éditer le script *setclasspath.sh* dans $CATALINA_HOME/bin et insérer les lignes suivantes :

   ```
   GLOBUS=/home1/globus/gt3
   export JAVA_HOME=$GLOBUS/java
   ```

   Cela permettra de pouvoir lancer tomcat depuis n'importe quelle machine.
4. rajouter dans le **.bashrc** :

   ```
   export CATALINA_HOME=$GLOBUS_LOCATION/tomcat
   ```
5. créer un lien symbolique **tomcat** sur cette version.

A.1.4 Axis

Installation de **Axis** :

1. télécharger **axis** ;
2. décompresser l'archive ;
3. créer un lien symbolique *axis* sur cette version ;
4. créer dans $CATALINA_HOME/webapps un lien symbolique *axis* sur le répertoire $GLOBUS_LOCATION/axis/webapps/axis ;
5. récuperer *xmlsec.jar* depuis http ://apache.crihan.fr/dist/xml/security/java-library/ puis le copier dans $GLOBUS_LOCATION/axis/webapps/axis/WEBINF/lib
6. mettre à jour la variable $CLASSPATH :

   ```
   export AXIS_HOME=$GLOBUS/axis
   export CLASSPATH=$CLASSPATH:$AXIS_HOME/lib/axis-ant.jar
   export CLASSPATH=$CLASSPATH:$AXIS_HOME/lib/axis.jar
   export CLASSPATH=$CLASSPATH:$AXIS_HOME/lib/jaxrpc.jar
   export CLASSPATH=$CLASSPATH:$AXIS_HOME/lib/commonsdiscovery.jar
   export CLASSPATH=$CLASSPATH:$AXIS_HOME/lib/commonslogging.jar
   export CLASSPATH=$CLASSPATH:$AXIS_HOME/lib/saaj.jar
   export CLASSPATH=$CLASSPATH:$AXIS_HOME/lib/wsdl4j.jar
   export CLASSPATH=$CLASSPATH:$AXIS_HOME/webapps/axis/WEBINF/lib/xerces.jar
   export CLASSPATH=$CLASSPATH:$AXIS_HOME/webapps/axis/WEBINF/lib/xmlsec.jar
   ```

A.1.5 ANT

Ant pourrait être comparé au célèbre outil make sous Unix. Il a été développé pour fournir un outil de construction indépendant de toute plate-forme car écrit avec et pour java. Pour une documentation plus complète, on pourra consulter le manuel à l'url suivante : http ://jakarta.apache.org/ant/manual/index.html

Installation du compilateur java **ant** :

1. télécharger **ant** ;
2. décompresser l'archive ;
3. exporter la variable d'environnement :

   ```
   export ANT_HOME=‘pwd‘/apache-ant-1.6.0
   ```

4. mise à jour de la variable **PATH** ;
5. télécharger le package d'expressions régulières pour ant : **jakarta-oro-2.0.8.jar** ;
6. décompresser et copier le jar dans le répertoire **$ANT_HOME/lib**.

A.2 L'installation

A.2.1 Exporter les variables d'environnement

Tout d'abord, il s'agit d'exporter les différentes variables d'environnement. Variables qui seront utiles au script d'installation durant la mise en place du produit.

```
export ANT_HOME=/usr/local1/apache-ant-1.6.1
export GLOBUS_LOCATION=/home1/globus/gt3
export GPT_LOCATION=/home1/globus/gt3
export PATH=$PATH:$ANT_HOME/bin
```

Notes :
– GLOBUS_LOCATION est le répertoire racine de Globus Toolkit.
– GPT_LOCATION est le répertoire racine du Grid Packaging Toolkit.

A.2.2 Installer Globus 3.0

Maintenant, nous rentrons dans le vif du sujet avec l'installation à proprement parlé du Globus Toolkit.

1. télécharger le **Globus Toolkit** ;
2. dézipper l'archive et lancer :

   ```
   ./install-gt3 $GLOBUS_LOCATION > gt3.log
   ```

   À noter que cette étape est relativement longue ;
3. copier le fichier **./BUILD/core-src/impl/java/endorsed/xalan.jar** dans le répertoire **$JAVA_HOME/jre/lib/endorsed** (si le répertoire endorsed n'existe pas, il faut le créer) ;
4. exécuter les commandes suivantes :

```
cd $GLOBUS_LOCATION
ant setup > setup.log
```

5. exécuter :

   ```
   $GLOBUS_LOCATION/setup/globus/setup-gsi
   ```

   Taper "**y**" pour continuer et "**q**" pour sauver les valeurs fournies ;

A.2.3 Demande de certificats

Principe des certificats

GLOBUS assure la sécurité de la grille grace à GSI. Le GSI vérifie la provenance d'une requête, l'authenticité d'un service, à l'aide de certificats électroniques cryptés par un algorithme asymétrique.

Le principe du cryptage asymétrique est le suivant :
Chaque utilisateur génère son couple de clé privée/publique, qui lui servira à être authentifié par le module de sécurité de la grille. La clef privée est gardée par l'utilisateur alors que la clef publique est distribuée à tous.
Si l'utilisateur crypte un message avec sa clef privée, tout le monde pourra le décrypter avec la clef publique et l'origine du message (utilisateur) sera garantie car seul l'utilisateur peut crypter un message avec sa clef privée.
Si une autre personne crypte un message avec la clef publique de l'utilisateur, seul celui-ci sera capable de le décrypter et donc de connaître le contenu.
Un double cryptage (clef privé de A/clef publique de B) permet alors de certifier l'origine du message (venant de A), message que seul B peut lire.

Dans GLOBUS :
- l'utilisateur génère son couple de clefs ;
- garde sa clef privée dans un endroit sécurisé ;
- envoie sa clef publique à l'autorité de certification (CA) ;
- le CA fabrique alors un certificat en cryptant avec sa propre clef privée :
  - le nom de la CA ;
  - le sujet (nom de l'utilisateur, organisme, ...) ;
  - la clef publique de l'utilisateur ;
  - les dates de validité.
- la CA renvoie le certificat à l'utilisateur.

A l'utilisation, le service de la grille (ex : gatekeeper) qui croit le CA (c-à-d : le service possède la clef publique du CA) en question, pourra décrypter le certificat de l'utilisateur et être sûr que la clef publique associée à l'utilisateur est la bonne. Le service demandera sous forme de challenge, à l'utilisateur de crypter un message pour voir si la clef privée détenue par l'utilisateur correspond à la clef publique certifiée par la CA. Si tout se passe bien, l'utilisateur est considéré comme identifié de façon sûre. (voir fig A.1)

Les machines constituant la grille utilisent le même système de certificat pour passer les challenges d'authentification.

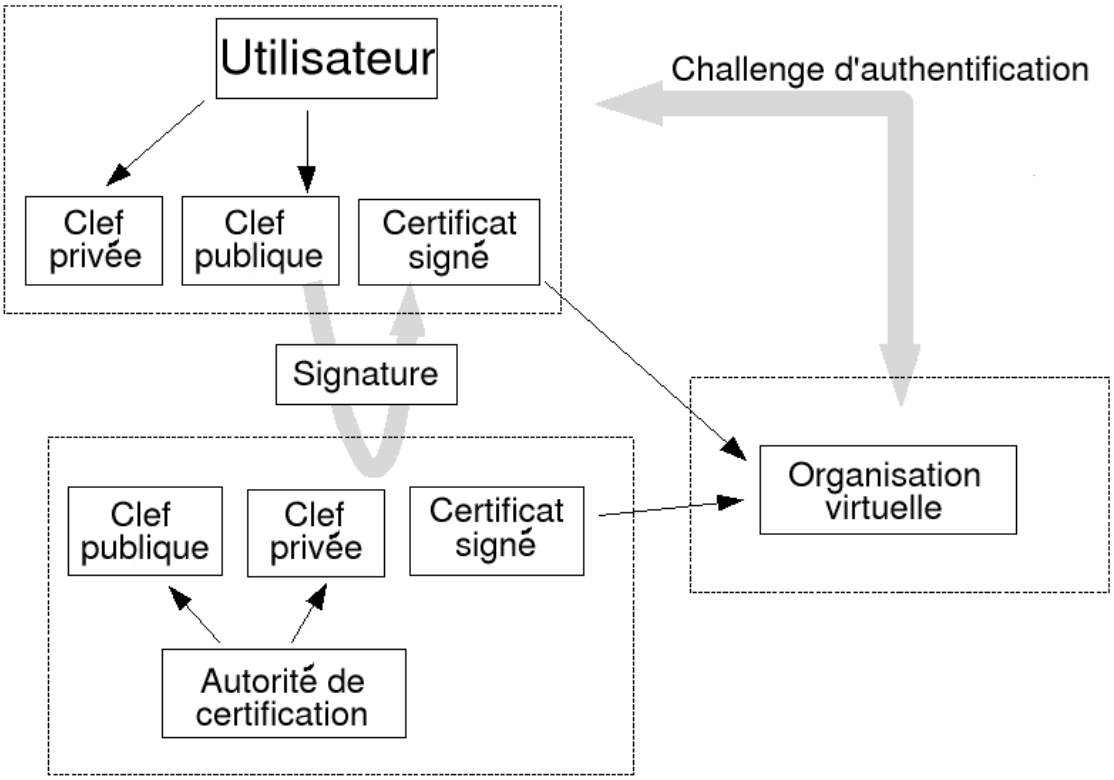

FIG. A.1 – *GSI : Principe des certificats.*

Obtenir un certificat hôte

Pour cela, j'ai suivi a peu de choses près le tutorial du site de globus à la page suivante servicecert.html.

Le protocole suivant récapitule les grandes étapes :
- télécharger l'archive globus_gcs_b38b4d8c_setup-0.1.tar.gz ;
- executer les commandes suivantes :

```
$GPT_LOCATION/sbin/gpt-build globus_gcs_b38b4d8c_setup-0.1.tar.gz
$GPT_LOCATION/sbin/gpt-postinstall
$GLOBUS_LOCATION/setup/globus_gcs_b38b4d8c_setup/setup-gsi
```

Créer si il n'existe pas le répertoire */etc/grid-security* pour stocker les certificats, puis executer la commande :

```
$GPT_LOCATION/bin/grid-cert-request -host hostname -dir /etc/grid-security
```

Utiliser le formulaire de la page précitée et mettre le résultat obtenue dans le fichier **/etc/grid-security/hostcert.pem** (le fichier doit avoir les droits suivants 644).

Obtenir un certificat utilisateur

```
source $GLOBUS_LOCATION/setenv.sh
$GLOBUS_LOCATION/bin/grid-cert-request
```

Utiliser le formulaire de la page suivante usercert.html et lui fournir le fichier obtenu par la commande précédente (**$HOME/.globus/usercert_request.pem**).

Le résultat obtenue doit être stocké dans le fichier **$HOME/.globus/usercert.pem** (le fichier doit avoir les droits suivants 644).

A.2.4 Vérification de l'installation

Avant de passer à l'étape suivante, on peut utiliser un script que j'ai trouvé sur un forum de discussion (Script de vérification) pour vérifier l'intégrité de notre installation.

A.2.5 Installation du composant GRAM de Globus 3.0

Exécuter les commandes suivantes :

```
./install-gt3-mmjfs $GLOBUS_LOCATION > gt3-mmjfs.log
$GLOBUS_LOCATION/bin/setperms.sh
```

Cela permettra de lancer grim (composant spécialisé dans la sécurité d'accès au GRAM) et mmjfs.

Ensuite, il faudra vérifier que les entrées correspondantes aux certificats ont bien été enregistrées dans le fichier **grid-mapfile** (fichier d'association entre la signature certifiée du futur client de la machine et un login existant).

Si ce n'est pas le cas, il suffit de l'éditer en mettant les deux informations suivantes :

- le sujet du certificat utilisateur (lancer en utilisateur **grid-cert-info -subject** et recopier la sortie) ;
- le login de l'utilisateur (résultat de **whoami**).

Créer ou éditer si il existe le fichier **/etc/grid-security/grim-port-type**, qui permettra de lister les services autorisés à s'exécuter au sein de la machine et insérer les lignes suivantes :

```
<?xml version= 1.0 ?>
<authorized_port_types>
  <port_type username="clement">
  http://www.globus.org/namespaces/managed_job/managed_job/ManagedJobPortType
  </port_type>
</authorized_port_types>
```

Si le fichier **grid-mapfile** contient beaucoup d'enregistrements, on pourra utiliser le script perl **gmap2gptx.pl** suivant :

Listing A.1 – *gmap2gptx.pl*

```
#!/bin/perl -w
#
# converts grid-mapfile into a grim-port-type.xml file
# prints unmatched lines to STDERR, grim-port-type.xml to STDOUT
```

```
# Marcus Christie, machrist@cs.indiana.edu

if (! exists $ARGV[0]) {
  print "Usage: gmap2gptx .../grid-mapfile > out.file\n";
  exit 1;
}

while (<>) {
  if ($_ =~ /".*"\s+(\w+)\s*$/) {
     $gptxEntry{$1} = "yes";
  } else {
     print STDERR "NO MATCH:";
     print STDERR;
  }
}

print "<authorized_port_types>\n";

foreach $key (sort keys %gptxEntry) {
  print "<port_type username=\"$key\">http://www.globus.org/
         namespaces/managed_job/managed_job/ManagedJobPortType
         </port_type>\n";
}

print "</authorized_port_types>\n";
```

Pour continuer nous allons sourcer les trois fichiers suivants :

```
source $GPT_LOCATION/etc/globus-user-env.sh
source $GPT_LOCATION/setenv.sh
source $GPT_LOCATION/bin/setperms.sh
```

et installer le gram :

```
$GLOBUS_LOCATION/setup/globus/setup-globus-gram-job-manager
```

Commenter alors les balises **containerHandlers** et **containerProxy** dans le fichier *serverconfig.wsdd*, fichier de configuration du serveur Globus :

```
<!--
<parameter name="containerHandlers"
value="org.globus.ogsa.impl.base.gram.mmjfs.UHERestartHandler
org.globus.ogsa.impl.security.grim.GrimContainerHandler"/>
-->
......
<!--
<parameter name="containerProxy" value="/tmp/containerProxy\_grim"/>
-->
```

Lancer le conteneur de grid service sur le port souhaité :

```
source $GLOBUS_LOCATION/etc/globus-user-env.sh
$GLOBUS_LOCATION/bin/globus-start-container -p 8060
```

Maintenant, il s'agit de lancer un client local qui émettra des requêtes RSL depuis le fichier fourni par Globus et situé dans **$GLOBUS_LOCATION/etc/test.xml**.

A.3 Lancer un client local

On émet un proxy client :

```
$GLOBUS_LOCATION/bin/grid-proxy-init
```

Cela produit le résultat suivant :

```
Your identity: /C=US/O=Globus Alliance/OU=User/CN=fb2fbba5cc.a2b516e2
Enter GRID pass phrase for this identity:
```

Ici on tape le mot de passe demandé à la génération d'une clé de certification user :

```
Creating proxy ........................... Done
Your proxy is valid until: Wed Mar 10 03:04:25 2004
```

On source les variables d'environnement de Globus :

```
source setenv.sh
export $GLOBUS_LOCATION=/opt/gt3
source $GLOBUS_LOCATION/etc/globus-user-env.sh
```

Pour être tranquille on insérera dans le .bashrc :

```
export $GLOBUS_LOCATION=/opt/gt3
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$GLOBUS_LOCATION/lib
export PATH=$PATH:$GLOBUS_LOCATION/bin
```

Maintenant, on teste un des services hébergés par le container, en l'occurence le GRAM :

```
bin/managed-job-globusrun -factory http://193.52.61.145:8080/ogsa/services/
base/gram/MasterForkManagedJobFactoryService -file etc/test.xml
```

Le fichier **test.xml** contient des requêtes au format RSL de demande d'exécution de tâche (eg /bin/echo ou /bin/date) avec des spécifications sur les ressources (quantité de CPU requise, nombre de processeurs, etc...)

Il reste à valider le résultat qui se trouve dans **/tmp/stdout** et de **/tmp/stderr** sans apparition de message d'erreur.

A.4 Post installation de Globus Toolkit 3.0

Nous allons maintenant procéder à la configuration des différents services de Globus.

A.4.1 GridFtp

GridFTP est un protocole à rendement élevé, sécurisé, fiable, permettant le transfert de données. Il est optimisé pour les réseaux étendus. Le protocole GridFTP est une adaptation du protocole FTP aux grilles de calcul.

Ce protocole est déjà installé et il ne nous reste plus qu'à le configurer. Il faut alors éditer le fichier **/etc/services** et rajouter le nom de service, le port et le protocole, comme suit :

```
gsiftp  2811/tcp
```

Ensuite, ajouter dans **/etc/inetd.conf**, en prenant soin de remplacer GLOBUS_LOCATION par le chemin de globus et PORT_MIN,PORT_MAX par la plage de numéros choisie :

```
gsiftp stream tcp nowait root /usr/bin/env env LD_LIBRARY_PATH=
GLOBUS_LOCATION/lib GLOBUS_TCP_PORT_RANGE=PORT_MIN,PORT_MAX GLOBUS_LOCATION/
sbin/in.ftpd -l -a -G GLOBUS_LOCATION
```

Puis relancer Inetd avec la commande : **killall -HUP inetd**.

Un test pour valider le service est :

```
globus-url-copy gsiftp://localhost:8060/tmp/file1 file:/tmp/file2
```

qui copie */tmp/file1* dans */tmp/file2*. On devra avant toute chose émettre un proxy client et remplacer *localhost* par le nom complet de la machine. On pourra, dans le cas où une erreur surviendrait, utiliser l'option **dbg** (debug) pour vérifier où la commande échoue.

A.4.2 Reliable Transfer Service : RFT

Ce service présent dans Globus 3.0 fournit des interfaces pour contrôler et monitorer les transferts entre serveurs GridFTP. Il s'agit d'une réadaptation de l'utilitaire *globus-url-copy* (encore présent sous GT3). Il nécessite **Postgresql** en plus de **GridFTP** et qu'ils soient lancés par exemple par respectivement :

```
export PGDATA=$GLOBUS_LOCATION/postgresql/pgsql/data
initdb
createdb ogsa
psql -d ogsa -f $GLOBUS_LOCATION/etc/rft_schema_ogsa.sql
$GLOBUS_LOCATION/sbin/in.ftpd -S [-p <port>]
```

A.4.3 MDS

MDS est un composant serveur qui se base sur l'implémentation open-source OpenLDAP de LDAP. Son utilisation nécessite l'installation d'un certificat ldap de façon similaire à celle des certificats hôtes et utilisateurs :

```
grid-cert-request -service ldap -host grappa.iut-bm.univ-fcomte.fr
```

Un fichier est automatiquement créé dans **/etc/grid-security/ldap/ldapcert_request.pem** qu'il faut envoyer à l'autorité de certification compétente et qui renvoie le fichier **ldapcert.pem** placé dans le même répertoire. On utilisera la même page que pour le certificat hôte (servicecert.html) en n'oubliant pas de choisir l'option **MDS Version 2 ("ldap" certificate)**.

Il faut changer ses droits (chmod 600 ldapcert.pem) et son appartenance remise au root (chown root ldapcert.pem).

Ensuite, on lance LDAP en mode root via **globus-mds start** et la vérification se fait par

```
$GLOBUS_LOCATION/bin/grid-info-search -anonymous -L
```

qui renvoie la configuration de l'hôte local.

A.4.4 GIIS

Le serveur GIIS demande une intervention au niveau de deux fichiers :

```
$GLOBUS_LOCATION/etc/grid-info-resource-register.conf
$GLOBUS_LOCATION/etc/grid-info-slapd.conf
```

Ce serveur demande de créer des programmes java pour exploiter ses potentialités.

A.4.5 GRIS

Nous allons essayé de démarrer manuellement le serveur GRIS, en utilisant la commande suivante :

```
$GLOBUS_LOCATION/sbin/SXXgris start
```

Si tout se déroule bien, la ligne suivante devrait apparaître :

```
Starting up Openldap 2.0 SLAPD server for the GRIS
```

À ce moment là, on modifiera le fichier */etc/init.d*, de manière à ce que le serveur GRIS démarre automatiquement.

Une fois le serveur démarré, on pourra effectuer des recherches, comme suit :

```
grid-info-search -anonymous -L
grid-info-search -anonymous -L -b 'Mds-Vo-name=site,o=grid'
```

Références

[AM03] Olivier Aumage and Guillaume Mercier. MPICH/MadIII : a Cluster of Clusters Enabled MPI Implementation. In *Proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, pages 26–35, Tokyo, May 2003. IEEE. 4.4.1

[Ama03] Abdelkader Amar. Environnement fonctionnel distribué et dynamique pour systèmes embarqués, Winter 2003.

[AMN01] Olivier Aumage, Guillaume Mercier, and Raymond Namyst. MPICH/Madeleine : a True Multi-Protocol MPI for High-Performance Networks. In *Proc. 15th International Parallel and Distributed Processing Symposium (IPDPS 2001)*, page 51, San Francisco, April 2001. IEEE. Extended proceedings in electronic form only. Extended version available as [**?**]. 4.4.1

[Bac03] Charles Bacon. Gt3 alpha admin guide, 2003. http ://www-unix.globus.org/ogsa/docs/alpha/admin/. A

[BAG00] Rajkumar Buyya, David Abramson, and Jonathan Giddy. Nimrod/g : An architecture for a resource management and scheduling system in a global computational grid, 2000. 4.4.4

[BDV94] Greg Burns, Raja Daoud, and James Vaigl. LAM : An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium*, pages 379–386, 1994. (document)

[BT89] Dimitri P. Bertsekas and John N. Tsitsiklis. Convergence rate and termination of asynchronous iterative algorithms. In *Proceedings of the 3rd international conference on Supercomputing*, pages 461–470. ACM Press, 1989. (document)

[Cha03] Thierry Charron. Le projet européen de grille de calcul (datagrid) : Concept de grilles de calcul. objectifs du projet et état d'avancement. Conservatoire National des Arts et Métiers, Mai 2003. 2.2, 2.4, 2.4.1, 2.4.2, 2.4.3, 2.4.4, 2.4.5

[DS04] Matt Davis and Rohit Sahasrabudhe. Speed-start your linux app 2004 : Installing globus toolkit 3.0 on linux for iseries and pseries, 04 January 2004. http ://www-106.ibm.com/developerworks/linux/library/l-ss4ip-globus.html. A

[Duc03] Frédéric Ducès. Grille de calcul - globus, 2003. 2.2

[ea02] Viktors Berstis et al. *Introduction to Grid Computing with Globus*. International Business Machines Corporation, December 2002. First Edition. (document), 3.1, 3.1, 3.4, 3.5, 3.5.1

[FGNC01] G. Fedak, C. Germain, V. N'eri, and F. Cappello. Xtremweb : A generic global computing system, 2001. (document)

[FK97] Ian Foster and Carl Kesselman. Globus : A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2) :115–128, Summer 1997. 3.1, 3.4

[FK99] Ian Foster and Carl Kesselman. The Globus project : a status report. *Future Generation Computer Systems*, 15(5–6) :607–621, 1999. (document), 3.2, 3.2, 3.3, 3.4

[FK00] Ian T. Foster and Carl Kesselman. Computational grids. In *VECPAR*, pages 3–37, 2000. 2.2

[FKT01] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid : Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150 :1–? ?, 2001. 2.2

[Fos00] Ian Foster. *The Data Grid : Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets*, volume 23, chapter n°3, pages 187–200. Journal of Network and Computer Application : Special issue on network based storage services, July 2000. (document)

[Fos02a] Ian Foster. The grid : A new infrastructure for 21st century science. *Physics Today*, pages 42–47, February 2002. (document)

[Fos02b] Ian Foster. The physiology of the grid - an open grid services architecture for distributed systems integration. *Open Grid Service Infrastructure WG, global grid forum*, 22 June 2002. (document)

[Fos02c] Ian Foster. What is the grid ? a three point checklist. *Grid Today*, 1(6), Summer 2002. (document)

[Fos03] Ian Foster. The grid : Computing without bounds. *Scientific American Inc.*, pages 78–85, April 2003. (document)

[Hed03] Misbah El Heddaji. Etude, mise en place et test d'une infrastructure de grille fondée sur les web services : Globus 3.0, 11 Septembre 2003. Université Pierre et Marie Curie, Formation Administrateur de Réseaux et Systèmes, EDF R&D - Sinetics - Groupe I22. A

[HJN01] Fabio Hernandez, Nicolas Jacq, and Sophie Nicoud. Datagrid, projet européen de grille de calcul, Septembre 2001. 2.1, 2.2

[IF99] Carl Kesselman (eds). Ian Foster. *The Grid : Blueprint for a new computing infrastructure*. Morgan Kauffman, 1999. (document)

[JBV04] Raphaël Couturier Jacques Bahi and Philippe Vuillemin. Asynchronous iterative algorithms for computational science on the grid : three case studies, 2004. LIFC, University of Franche-Comté, France. (document), 4.1.1, 4.1.2, 4.1.3

[LF03] Bart Jacob *et al* Luis Ferreira. *Globus Toolkit 3.0 Quick Start*. IBM, September 2003. A

[NNTHG02] A. Natrajan, A. Nguyen-Tuong, M. Humphrey, and A. Grimshaw. The legion grid portal, 2002. (document)

[Rom03] David Romaric. Une grille de calcul pour la recherche. Université Louis Pasteur, Strasbourg, 12 octobre 2003. (document)

[Sag] Christian Saguez. Infrastructures grille. (document)

[SL03] Jeffrey M. Squyres and Andrew Lumsdaine. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, number 2840 in Lecture Notes in Computer Science, pages 379–387, Venice, Italy, September / October 2003. Springer-Verlag. (document)

[Sot03] Borja Sotomayor. Gt3 alpha2/3 installation (for redhat 7.x), 2003. http ://www.cs.binghamton.edu/ xling/gt3-alpha-installation.html. A

[Sou03] Toni Soueid. Grille de calcul : Etat de l'art. Telecom Paris - Ecole Nationale Supérieure des Télécommunications, June 2003. 3, 3.1, 3.2, 3.3, 3.4, 3.5, 3.5.1, 3.6

[Vui03] Philippe Vuillemin. Algorithmique itérative asynchrone pour le calcul sur grille avec un environnement de programmation orienté objet, September 2003. Mémoire de DEA. 4.1.3, 4.6

Sites internet

- Site central de Globus
  http ://www.globus.org
- Site de développement d'IBM, avec de nombreux tutoriaux
  http ://www-136.ibm.com/developerworks
- Page personnelle de Borja Sotomayor, avec un tutorial plutôt complet
  http ://www.casa-sotomayor.net/gt3-tutorial
- Documentations de l'INRIA sur Globus 2.2 liées à l'application MECAGRID
  http ://www-sop.inria.fr/smash/mecagrid/public/index.htm
- Description du Globus Toolkit sur le site de l'ENS Lyon
  http ://graal.ens-lyon.fr/ fsuter/pages/Session%204/index.htm
- Page personnelle de Ian Foster, avec de nombreux liens, notamment vers ses articles
  http ://www-fp.mcs.anl.gov/ foster/
- Page personnelle de Karl Kesselman, avec quelques liens et la liste de ses publications dans son curriculum vitae
  http ://www.isi.edu/ carl
- Site central de LAM/MPI
  http ://www.lam-mpi.org/
- Site central de Legion
  http ://www.legion.virginia.edu

Résumé

Dans un contexte de calcul sur grille, il existe plusieurs middleware, tel que Legion, Globus, Harness ou encore XtremWeb, qui présentent des boîtes à outils permettant de commencer rapidement à faire du calcul sur grille.

Au cours de ce stage, nous utilisons **Globus** afin de le tester et de savoir si, à long terme, il pourrait être utile aux calculs scientifiques menés au sein de l'équipe Algorithmique Numérique Distribuée de Belfort.

Nous portons sous Globus une application MPI modélisant un mécanisme de cinétique chimique d'advection-diffusion avec une version synchrone et l'autre asynchrone.

Nous effectuons une série de tests démontrant que les performances de Globus sont très proches de celles de MPI et démontrant l'efficacité de l'asynchronisme sur le synchronisme dans le contexte du calcul sur grille.

Mots clefs : Globus, Middleware, Asynchronisme, MPI.

Abstract

In a grid computing context, there exist several middleware, such as Legion, Globus, Harness or XtremWeb, which present toolboxes making it possible to quickly start grid computing.

During this training course, we use **Globus** in order to test it and to know if, in the long run, it could be useful for the scientific computations carried out in the Algorithm Distributed Numerical team of Belfort.

We deploy under Globus a MPI application modelling a chemical mechanism of kinetics of advection-diffusion with synchronous and asynchronous versions.

We make a series of tests showing that Globus has results close to those of MPI and showing the effectiveness of asynchronism on synchronism in the context of grid computing.

Keywords : Globus, Middleware, Asynchronism, MPI.