Numéro d'ordre : 3216

THÈSE

présentée devant

L'UNIVERSITÉ DE RENNES 1

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1
Mention INFORMATIQUE

par

Sébastien LACOUR

Équipe d'accueil : projet PARIS, IRISA, INRIA Rennes

École doctorale : MATISSE

Composante universitaire : IFSIC

Contribution à l'automatisation du déploiement d'applications sur des grilles de calcul

soutenue le 8 décembre 2005 devant la Commision d'Examen

Composition du jury

M. Franck CAPPELLO, directeur de recherche	rapporteur
M. Denis CAROMEL, professeur	examinateur
M. Frédéric DESPREZ, directeur de recherche	examinateur
M. Ian FOSTER, professeur	rapporteur
M. Christian PÉREZ, chargé de recherche	co-encadrant de thèse
M. Thierry PRIOL, directeur de recherche	directeur de thèse

Table des matières

Table des figures

Liste des définitions

Chapitre 1

Introduction

LE CALCUL SCIENTIFIQUE est un champ d'application important de l'informatique, notamment en ce qui concerne la simulation numérique. Par exemple, « *El Niño* » est un phénomène météorologique[1] encore mal compris. Les météorologistes du monde entier poursuivent des recherches pour expliquer ce phénomène en le simulant par des modèles couplés océan-atmosphère. D'autres exemples existent dans le domaine de la physique : grâce à la simulation numérique, les physiciens ont pu faire d'importantes avancées dans la compréhension théorique de phénomènes tels que la fusion nucléaire [31].

Plusieurs raisons peuvent mener au recours à la simulation numérique.

Sécurité : le réglage des paramètres de fonctionnement des cœurs de réacteurs nucléaires ou les expériences de fusion nucléaire dans le réacteur expérimental ITER (*International Thermonuclear Experimental Reactor*, [198]) présentent des *dangers* importants, et doivent faire l'objet de simulations numériques au préalable. L'énergie qui régira le plasma dans le réacteur ITER sera essentiellement d'origine interne au plasma (réactions de fusion nucléaire), et non contrôlée de manière externe, car les plasmas sont très loin de l'équilibre thermique. Il sera donc inévitable de conduire des simulations précises avant de se lancer dans des expériences réelles grandeur nature [31].

Intérêt économique : la déformation des structures des automobiles et les tests lors de la conception de nouveaux avions sont *plus onéreux à réaliser physiquement qu'à modéliser et simuler* sur ordinateur. Par ailleurs, le réacteur ITER devrait coûter environ un million de dollars par jour de fonctionnement [198] : avoir une idée des résultats attendus et savoir si les effets attendus peuvent être observables permettra de ne pas conduire des expériences pour rien. Les études de faisabilité d'une expérience dans un tel appareil dureront plusieurs mois, et seront fondées sur des simulations numériques.

Impossibilité physique : les expériences pour comprendre la dynamique des protéines ou encore le comportement des particules infiniment petites (quarks) en physique quantique peuvent très difficilement être contrôlées ou nécessitent des quantités d'énergie que *l'homme ne sait pas encore produire*.

Ainsi, le calcul scientifique intervient dans de nombreux domaines : astrophysique (simulation de l'explosion d'une supernova, collision de trous noirs), automobile et aérospatiale,

[1] Inversion du sens des vents et des courants marins dans le sud de l'océan Pacifique, entraînant des catastrophes humaines et écologiques.

modélisation climatique (simulation de tornades, prévision du climat de la Terre au siècle prochain), économie (modélisation de l'économie mondiale), *etc.*

Complexité des applications

Pour effectuer des simulations numériques toujours plus *réalistes* et plus *précises*, le calcul scientifique s'est orienté à la fois vers le *calcul intensif* et le *couplage de codes*.

La *précision* dans les simulations numériques s'obtient par la finesse du maillage (spatial) et la diminution du pas de temps. Cette augmentation de la précision des simulations nécessite de pouvoir supporter des *calculs intensifs* sur des volumes de données importants.

De plus, la compréhension des *phénomènes individuels* (diffusion d'énergie, transport de matière, thermodynamique, électromagnétisme, réactions nucléaires, mécanique quantique, *etc.*) qui régissent le comportement d'un plasma ne suffit pas. Le comportement du plasma est le fruit du *couplage* de tous ces phénomènes physiques : ce couplage est beaucoup plus riche d'enseignements que la somme des phénomènes individuels [31]. Il en est de même pour la simulation du comportement de systèmes complexes tels qu'un satellite, un avion ou un réacteur nucléaire.

Parmi les grandes classes d'applications de calcul scientifique qui sont employées pour mettre en œuvre des simulations complexes et numériquement intensives, nous comptons les applications distribuées (à base de composants), les applications parallèles (par passage de messages et par mémoire virtuellement partagée), ainsi que les applications mixtes, constituées de composants qui contiennent des programmes parallèles. Ainsi, les applications de calcul scientifique sont souvent complexes, tant par leur taille que par leur structure. De plus, elles possèdent toutes des mécanismes de déploiement distincts, souvent non spécifiés.

Puissance et complexité des grilles de calcul

Le calcul scientifique fait souvent intervenir du *calcul numérique intensif*, sur des *problèmes de grande taille*. Il nécessite donc une importante puissance de calcul et de stockage. Les *grilles informatiques* sont une réalité tangible qui est à même de satisfaire les besoins importants de calcul et de stockage des applications, grâce notamment à l'amélioration des performances des réseaux longue distance.

Cependant, les grilles de calcul sont des environnements très *complexes* : elles sont composées de ressources hétérogènes (tant les machines que les réseaux de communication), distribuées à travers plusieurs pays, et elles appartiennent à des instituts qui se font une confiance limitée. Les intergiciels d'accès aux grilles résolvent des problèmes de sécurité : chiffrement des données pour la confidentialité, authentification des ressources et des utilisateurs, autorisation d'accès, *etc.* Ils permettent également de soumettre des tâches et de transférer des données sur des machines distantes. Cependant, ces intergiciels restent d'un usage complexe, et fournissent des services d'assez bas niveau, par rapport au déploiement d'applications.

Problématique

Actuellement, les applications de calcul scientifique sont particulièrement difficiles à déployer sur des grilles. Le processus de déploiement est encore trop *manuel* et *techniquement complexe*. L'utilisateur qui veut déployer son application doit faire preuve d'expertise dans le domaine du lancement des différents types d'applications, ainsi qu'en matière d'intergiciels d'accès aux grilles de calcul. De plus, l'utilisateur doit lui-même connaître les besoins de l'application, découvrir et sélectionner les ressources d'exécution nécessaires à l'application, choisir les implémentations de l'application pour chaque ressource sélectionnée, réaliser tous les transferts de fichiers et les lancements de tâches à distance, puis configurer l'application.

Objectif

Notre défendons la thèse selon laquelle il est possible aux scientifiques d'*accéder simplement à une grande puissance informatique pour y lancer des applications complexes de calcul scientifique*, sans être experts en informatique ou en grilles. Notre objectif est de masquer aux utilisateurs les difficultés du déploiement liées aux applications et aux grilles. Pour atteindre cet objectif, nous montrons comment le déploiement d'applications peut être *automatisé* dans un environnement de grilles de calcul.

Cet objectif va dans le sens de la définition de l'informatique comme la science du traitement *automatique* de l'information, et il rejoint l'un des objectifs initiaux des grilles de calcul, à savoir la *transparence d'utilisation*.

Organisation de ce document et contributions

La première partie de ce document présente le contexte de nos travaux :

- le chapitre 2 présente quelques classes d'*applications utilisées pour le calcul scientifique* : les applications distribuées, les applications parallèles, et les applications mixtes, ainsi que quelques exemples de technologies qui implémentent ces modèles de programmation ;
- le chapitre 3 définit ce que nous appelons *grilles de calcul*, puis il montre qu'elles sont des infrastructures d'exécution intéressantes pour les applications que nous visons ;
- le chapitre 4 définit ce que nous entendons par « déploiement d'applications », et montre quelles sont les *difficultés actuelles à déployer des applications* complexes de calcul scientifique sur des grilles informatiques ; nous en déduisons la nécessité d'*automatiser* le processus de déploiement.

La deuxième partie détaille nos contributions en matière d'automatisation du déploiement d'applications de calcul scientifique sur des grilles :

- le chapitre 5 présente l'*architecture générale de notre modèle de déploiement automatique* d'applications sur des grilles de calcul ;
- le chapitre 6 présente notre modèle de *description de la topologie réseau* complexe des grilles de calcul, pour permettre à l'outil de déploiement de placer les applications en connaissance de cause ;

- le chapitre 7 présente nos modèles de *description spécifique* et de *packaging d'applications* parallèles MPI et d'applications mixtes GRIDCCM, en vue de l'automatisation de leur déploiement : ces formalismes de description sont indépendants de toute infrastructure d'exécution ;
- le chapitre 8 introduit notre modèle de *description générique d'applications*, qui permet à l'outil de déploiement de planifier le lancement de n'importe quel type d'application, après conversion de sa description spécifique en description générique ;
- et le chapitre 9 montre comment l'outil de déploiement peut configurer automatiquement des applications parallèles pour leur permettre de s'adapter à la hiérarchie des performances de communication dans les grilles de calcul.

La troisième partie traite des questions de mise en œuvre et d'évaluation de performances de nos résultats :

- le chapitre 10 détaille l'implémentation d'ADAGE, notre outil de déploiement automatique qui met en œuvre et valide les contributions présentées dans la deuxième partie de ce document ;
- le chapitre 11 présente une évaluation des gains de performance apportés par l'implémentation hiérarchique d'un protocole de cohérence mémoire au sein du système de MVP DSM-PM2.

Enfin, dans le dernier chapitre, nous tirons les conclusions de nos résultats, puis nous traçons un aperçu des prolongements qui s'inscrivent dans la perspective de nos travaux.

Publications et présentations

Les résultats que nous présentons dans ce document ont fait l'objet de diverses publications et présentations. Elles concernent :

- l'architecture générale de notre modèle de déploiement automatique d'applications [6], présentée à DECOR'2004 [G] ;
- la description de la topologie des réseaux des grilles de calcul dans un article (« *full paper* » [4]) présenté à GRID2004 [A], avec un taux d'acceptation de 23% ;
- la description spécifique et le packaging d'applications MPI [11] ;
- la description générique d'applications dans un article (« *short paper* » [5]) présenté à GRID2005 [C], avec un taux d'acceptation de 14% (après 19% de « *full papers* ») ;
- une expérience réussie de déploiement automatique d'applications CCM via Globus [3], présentée à CD'2004 [E] ;
- une expérience réussie de déploiement automatique d'applications MPICH-G2 présentée à SIAM-CSE'2005 [B] ;
- l'implémentation hiérarchique des opérations collectives MPI et l'accès à la topologie réseau sous-jacente dans MPICH-G2 [12, 8, 9], écrits en collaboration internationale, et présentés à SIAM-PP'2004 [F] ;
- l'implémentation hiérarchique des verrous de synchronisation dans le système de MVP DSM-PM2 [2, 10], présentée à DSM'2003 [D].

Ces différentes publications ont en outre fait l'objet de rapports de recherche INRIA et de publications internes IRISA.

Chapitre de livres

[1] Alexandre DENIS, Sébastien LACOUR, Christian PÉREZ, Thierry PRIOL et André RIBES. *Engineering the Grid: status and perspective*, chapitre Programming the Grid with components : models and runtime issues. American Scientific Publishers, 2005. Édité par Beniamino DI MARTINO, Jack DONGARRA, Adolfy HOISIE, Laurence T. YANG et Hans ZIMA.

Conférences internationales avec comité de lecture

[2] Gabriel ANTONIU, Luc BOUGÉ et Sébastien LACOUR. Making a DSM Consistency Protocol Hierarchy-Aware: an Efficient Synchronization Scheme. *In 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2003), Workshop on Distributed Shared Memory on Clusters (DSM 2003)*, pages 516–523. IEEE TFCC, Tokyo, Japon, mai 2003.

[3] Sébastien LACOUR, Christian PÉREZ et Thierry PRIOL. Deploying CORBA Components on a Computational Grid: General Principles and Early Experiments Using the Globus Toolkit. *In 2nd International Working Conference on Component Deployment (CD 2004)*, édité par Wolfgang EMMERICH et Alexander L. WOLF, volume 3083 de *LNCS*, pages 35–49. Springer-Verlag, Édimbourg, Écosse, UK, mai 2004.

[4] Sébastien LACOUR, Christian PÉREZ et Thierry PRIOL. A Network Topology Description Model for Grid Application Deployment. *In 5th IEEE/ACM International Workshop on Grid Computing (Grid2004)*, édité par Rajkumar BUYYA, pages 61–68. Pittsburgh, PA, USA, novembre 2004.

[5] Sébastien LACOUR, Christian PÉREZ et Thierry PRIOL. Generic Application Description Model: Toward Automatic Deployment of Applications on Computational Grids. *In 6th IEEE/ACM International Workshop on Grid Computing (Grid2005)*. Springer-Verlag, Seattle, WA, USA, novembre 2005.

Conférence nationale avec comité de lecture

[6] Sébastien LACOUR, Christian PÉREZ et Thierry PRIOL. A Software Architecture for Automatic Deployment of CORBA Components Using Grid Technologies. *In 1ère Conférence Francophone sur le Déploiement et la (Re)Configuration de Logiciels (DECOR'2004)*, pages 187–192. Grenoble, France, octobre 2004.

Actes d'école sans comité de lecture

[7] Alexandre DENIS, Sébastien LACOUR, Christian PÉREZ, Thierry PRIOL et André RIBES. Composants logiciels et grilles de calcul. *In Actes de l'École thématique sur la DistRibUtIon de Données à grande Échelle (DRUIDE 2004)*, pages 11–22. Domaine de Port-aux-Rocs, Le Croisic, France, mai 2004.

Rapports de recherche

[8] Nicholas T. KARONIS, Bronis de SUPINSKI, Ian FOSTER, William GROPP, Ewing LUSK et Sébastien LACOUR. A Multilevel Approach to Topology-Aware Collective Operations

in Computational Grids. Rapport technique ANL/MCS-P948-0402, Mathematics and Computer Science Division, Argonne National Laboratory, USA, avril 2002.

[9] Sébastien LACOUR. *MPICH-G2 collective operations: performance evaluation, optimizations*. Rapport de stage MIM2, Magistère d'informatique et modélisation (MIM), ENS Lyon, France, Mathematics and Computer Science Division, Argonne National Laboratory, USA, septembre 2001.

[10] Sébastien LACOUR. *Mémoire virtuellement partagée pour grappes de grappes : conception, mise en œuvre et évaluation d'un protocole de cohérence*. Rapport de stage de DEA, DEA d'informatique de l'IFSIC, Université de Rennes 1, France, juin 2002.

[11] Sébastien LACOUR, Christian PÉREZ et Thierry PRIOL. Description and Packaging of MPI Applications for Automatic Deployment on Computational Grids. Rapport de recherche RR-5582, INRIA, IRISA, Rennes, France, mai 2005. URL : `http://www.inria.fr/rrrt/rr-5582.html`.

[12] James D. TERESCO, Joseph E. FLAHERTY, Scott B. BADEN, Jamal FAIK, Sébastien LACOUR, Manish PARASHAR, Valerie E. TAYLOR et Carlos A. VARELA. Approaches to Architecture-Aware Parallel Scientific Computation. Rapport de recherche CS-04-09, Williams College Department of Computer Science, Williamstown, MA, USA, 2005. Publié sous *Proceedings of the 11th Conference on Parallel Processing for Scientific Computing of the Society for Industrial and Applied Mathematics (SIAM-PP2004) : Frontiers of Scientific Computing*.

Présentations internationales

[A] A Network Topology Description Model for Grid Application Deployment. Présentation, novembre 2004. 5th IEEE/ACM International Workshop on Grid Computing (Grid 2004), Pittsburgh, PA, USA.

[B] Automatic Deployment of MPI applications on a Computational Grid. Présentation, février 2005. Conference on Computational Science and Engineering of the Society for Industrial and Applied Mathematics (SIAM-CSE'2005), Orlando, FL, USA.

[C] Generic Application Description Model: Toward Automatic Deployment of Applications on Computational Grids. Présentation, novembre 2005. 6th IEEE/ACM International Workshop on Grid Computing (Grid 2005), Seattle, WA, USA.

[D] Making a DSM Consistency Protocol Hierarchy-Aware: an Efficient Synchronization Scheme. Présentation, mai 2003. Workshop on Distributed Shared Memory on Clusters (DSM 2003), Tokyo, Japon.

[E] Deploying CORBA Components on a Computational Grid: General Principles and Experiments Using the Globus Toolkit. Présentation, mai 2004. 2nd International Working Conference on Component Deployment (CD 2004), Édimbourg, Écosse, UK.

[F] A Multi-level Topology-Aware Approach to Implementing MPI Collective Operations for Computational Grids. Présentation, février 2004. 11th Conference on Parallel Processing for Scientific Computing of the Society for Industrial and Applied Mathematics (SIAM-PP'2004), San Francisco, CA, USA.

Présentation nationale

[G] A Software Architecture for Automatic Deployment of CORBA Components Using Grid Technologies. Présentation, octobre 2004. 1ère Conférence Francophone sur le Déploiement et la (Re)Configuration de Logiciels (DECOR'2004), Grenoble, France.

Première partie

Contexte d'étude

Chapitre 2

Applications concurrentes pour le calcul scientifique

Sommaire

L'objectif de ce chapitre est de présenter trois grandes classes d'applications concurrentes qui sont utilisées pour le calcul scientifique, et sur le déploiement desquelles ce travail se focalise. Après une rapide mise en place de quelques définitions générales concernant les applications concurrentes (section 2.1), ce chapitre présente d'abord le modèle d'application distribuée (section 2.2), puis le modèle d'application parallèle (section 2.3), et enfin le modèle d'application mixte (section 2.4). Les applications que nous qualifions de « mixtes » combinent à la fois les aspects des applications distribuées et ceux des applications parallèles. Pour chacune de ces trois grandes classes d'applications, nous décrivons un ou deux exemples de technologies qui mettent en œuvre le modèle.

2.1 Définitions

L'entité la plus simple que nous considérons est le « programme », qui correspond à un « code source ». Un programme peut être compilé pour donner un « exécutable ». À un programme donné peuvent correspondre plusieurs exécutables pour différentes plates-formes d'exécution. Une plate-forme d'exécution est caractérisée par un système d'exploitation, une architecture matérielle, et un ensemble de bibliothèques disponibles.

Un processus est un programme exécutable en cours d'exécution sur un ordinateur. Un thread est un flot d'exécution au sein d'un processus : tous les threads d'un processus partagent l'espace d'adressage virtuel du processus. Un processus est constitué d'au moins un thread.

Définition 2.1 : application — *Une application est un ensemble de programmes qui visent à la résolution d'un problème commun.*

Suivant cette définition, un programme résout un sous-problème de l'application, et tous les programmes qui constituent l'application *coopèrent* pour résoudre un même problème. Une application peut éventuellement être réduite à un seul programme.

Une application concurrente peut être définie de la manière suivante.

Définition 2.2 : application concurrente — *Une application est dite concurrente lorsqu'elle est constituée de plusieurs flots de contrôle qui s'exécutent simultanément (à un moment donné) et qui coopèrent.*

Il peut s'agir d'un seul processus qui contient plusieurs threads (application faite d'un seul programme), ou bien de plusieurs processus (application faite de plusieurs programmes), qui s'exécutent en même temps et coopèrent. La *simultanéité* s'entend à l'échelle du temps d'exécution de l'application, et non à l'échelle du temps d'exécution d'une instruction machine par un processeur : les intervalles de temps d'exécution des différents flots de contrôle d'une application concurrente se recouvrent. La notion de *coopération* signifie que les différents flots de contrôle d'une application concurrente doivent poursuivre un objectif commun (la résolution du même problème) et s'échanger du contrôle et/ou des données.

Les sections suivantes présentent trois classes d'applications concurrentes : les applications distribuées, les applications parallèles, et les applications mixtes. Il est important de noter que nous ne prétendons pas couvrir *tout le spectre* des modèles d'applications concurrentes telles que nous venons de les définir. Nous ne couvrons pas non plus l'ensemble des classes d'applications utilisées par le calcul scientifique. Simplement nous considérons un ensemble significatif de classes d'applications concurrentes qui présentent des propriétés intéressantes (décrites dans ce chapitre), et qui sont effectivement utilisées pour le calcul scientifique.

2.2 Applications distribuées

Cette section présente la classe des applications distribuées, qui sont utilisées dans le calcul scientifique. Dans cette classe d'applications concurrentes, nous nous focalisons sur les applications à base de composants logiciels, dont nous détaillons les avantages. Ensuite, cette section introduit le modèle de composants CORBA sur le déploiement duquel nous avons travaillé.

2.2.1 Programmation distribuée

2.2.1.1 Propriétés générales

L'archétype de la programmation distribuée est l'application « client-serveur », dans laquelle un programme, appelé « client », demande à un autre programme, nommé « serveur », d'exécuter une tâche appelée « service ». Par exemple, lorsqu'un individu veut accéder à son compte bancaire depuis Internet, son navigateur joue le rôle de client, et le site de la banque fait office de serveur pour répondre aux requêtes du client.

> « *A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.* » Leslie Lamport

Les applications distribuées comportent le plus souvent les caractéristiques suivantes.

Dynamicité. Dans une application distribuée typique, le serveur est en attente des requêtes des clients pour activer et exécuter un service. Une fois le service rendu, le client se déconnecte du serveur.

Hétérogénéité. Le plus souvent, les clients et les serveurs s'exécutent sur des ordinateurs distincts, et donc sur des plates-formes avec des systèmes d'exploitation et des architectures matérielles différents.

Sécurité. *A priori*, les serveurs ne connaissent pas par avance les clients qui sont susceptibles de se connecter à eux pour y requérir un service, et les applications distribuées peuvent s'exécuter dans un milieu partagé par n'importe quel utilisateur (tel qu'Internet). Il peut donc être utile de sécuriser les échanges entre clients et serveurs.

2.2.1.2 Modèles de programmation et de communication

La programmation distribuée a été un champ de recherche très actif qui a débouché sur plusieurs modèles de programmation. Cette section résume les différents modèles de programmation distribuée au travers de leurs modèles de communication.

Par définition, un *modèle de programmation* régit la conception et la structuration d'un programme ou d'une application. Un *modèle de communication* définit les méthodes d'échange d'information entre les flots d'exécution d'un programme ou d'une application.

Le *passage de messages* et le *passage de documents* sont deux modèles de programmation distribuée où les *communications* sont explicites, et qui n'imposent pas de structuration de l'application. Le passage de messages consiste à *établir un canal de communication* entre deux flots d'exécution pour y *envoyer* et *recevoir* des données, qui sont les deux opérations fondamentales de ce modèle. Le passage de documents est un cas particulier de plus haut niveau du passage de messages où les données sont *typées*. Les primitives de base sont *envoyer* et *attendre* un document, et le type du document détermine sa syntaxe pour son destinataire. Ce modèle est indépendant des protocoles de communication, et il est notamment utilisé par les *Web Services* [160].

L'*appel de procédure à distance* et l'*appel de méthode à distance* sont deux modèles de programmation distribuée où les *traitements* sont mis au premier plan, et les communications sont implicites. L'appel de procédure à distance (RPC, ou *Remote Procedure Call*) consiste à réaliser des appels de procédure entre des programmes distribués. Les paramètres des procédures

permettent d'échanger des données entre l'appelant et l'appelé. Ce modèle permet d'appeler une fonction qui s'exécute à distance comme si elle s'exécutait localement (*i.e.*, dans le même processus). L'appel de méthode à distance (RMI, ou *Remote Method Invocation*) consiste à réaliser des appels de méthodes sur des objets distants. L'appel de méthode à distance est la transposition du mécanisme de RPC à la programmation orientée objet : les RMI ajoutent les avantages de l'approche orientée objet au modèle d'appel de procédure à distance. Les implémentations les plus connues sont les JAVA RMI [92] et CORBA (*Common Object Request Broker Architecture*, [53]).

Dans tous les cas, ces modèles de programmation se focalisent sur la structuration de l'application ou bien sur les méthodes d'échange de données entre les tâches de l'application. *Aucun de ces paradigmes ne prévoit comment lancer les applications*, car ils sont nés à une époque où les applications étaient suffisamment simples pour que les individus qui développaient une application et ceux qui la déployaient soient exactement les mêmes. De nos jours, les applications sont devenues plus complexes pour obtenir des simulations numériques plus réalistes, et les anciennes méthodes *ad hoc* de lancement d'applications ne sont plus adaptées. Afin de résoudre ces problèmes de structuration et de déploiement des applications, un nouveau modèle de programmation est apparu, comme l'illustre la section suivante.

2.2.2 Programmation par composants logiciels

2.2.2.1 Des objets distribués aux composants logiciels

Les objets distribués constituent un mécanisme d'abstraction très puissant : l'approche orientée objet a largement démontré ses avantages dans de nombreuses applications [169], notamment lors de la phase de conception des applications. Cependant, l'approche orientée objet n'a pas atteint tous ses objectifs. Par exemple, il est très difficile de trouver les dépendances d'un objet dans une application qui implique un grand nombre d'objets et qui utilise l'héritage et la délégation. Les dépendances entre les objets ne sont pas très visibles : ces informations sont enfouies dans le code source. La réutilisabilité et la maintenance du code ne sont, en pratique, pas satisfaisantes : on ne peut pas simplement extraire quelques objets d'une application pour les réutiliser dans une autre application. Pour les applications distribuées, l'approche orientée objet ne fournit aucun support pour le déploiement d'applications ni pour la connexion entre objets. Par exemple, ni JAVA RMI [92], ni CORBA 2 [53] ne spécifient comment installer un objet à distance.

Tandis que l'approche orientée objet structure la conception d'applications, les composants logiciels [151] mettent en valeur la composition (assemblage et connexion) et le déploiement de composants. Ainsi, les composants ne sont pas directement liés à un modèle de communication précis. Les modèles de composants peuvent être fondés sur le passage de documents, l'appel de procédure à distance (DCOM[1] de Microsoft [175]) ou encore l'appel de méthode (CCM, CORBA *Component Model*, [51] de l'OMG[2]).

[1] *Distributed Component Object Model*.

[2] *Object Management Group*, un consortium académique et industriel de normalisation en informatique.

2.2.2.2 Notion de composants logiciels

Parmi les multiples définitions possibles des composants logiciels, celle de Clemens SZYPERSKI [151] est la plus utilisée :

> **Définition 2.3 : composant logiciel —** *Un composant logiciel est une unité de composition qui spécifie contractuellement ses interfaces et qui définit explicitement toutes ses dépendances de contexte. Un composant logiciel peut être déployé indépendamment et est sujet à composition par une tierce partie.*

D'une part, l'opération principale d'un composant est la *composition* avec d'autres composants : les composants disposent de « ports », qui sont les points de connexion, et donc de communication, entre les composants. Chaque port est typé par une interface, et un composant peut soit *fournir*, soit *utiliser* cette interface. On peut donc connecter le port d'un composant qui utilise une interface au port d'un autre composant qui fournit une interface du *même type*. Ainsi, la composition est effectuée via des interfaces bien spécifiées.

D'autre part, l'aspect de déploiement des composants fait partie intégrante de leur définition : un composant logiciel doit pouvoir être déployé indépendamment des autres composants sur des machines distribuées. Ainsi, toutes les informations sur les exécutables du composant, ses dépendances vis-à-vis de bibliothèques, ses interfaces de connexions, *etc.* sont spécifiées.

2.2.2.3 Intérêts des composants logiciels pour le calcul scientifique

Construction d'applications par assemblage. Les applications à base de composants logiciels se construisent par *assemblage* de composants (éventuellement déjà existants), et non par *développement* (ou programmation) en écrivant du code source monolithique. Cette modularité dans la construction d'applications permet de composer des applications complexes en un temps réduit grâce à la réutilisation de composants déjà existants.

La simulation numérique est une classe importante du calcul scientifique. De plus en plus, elle se réalise par couplage de codes : par exemple, pour une application complexe de couplage multi-physique, différents codes peuvent simuler des physiques différentes. Chaque code, qui a été écrit par un spécialiste du domaine, peut être encapsulé dans un composant, c'est-à-dire une « boîte noire » (que le code source soit propriétaire ou non) avec une interface publique. Ainsi, le spécialiste qui écrit le code d'un composant peut focaliser son expertise sur son domaine, ce qui permet d'améliorer la qualité des logiciels.

À titre d'exemple, le projet ACI GRID[3] HydroGrid [193] a pour but de modéliser et simuler des transferts de fluides et de solutés dans des milieux géologiques souterrains, afin de résoudre les problèmes d'intrusion d'eau salée dans les aquifères ou de stockage profond des déchets nucléaires. Ces problèmes font intervenir des phénomènes physico-chimiques complexes et variés. Chaque phénomène physico-chimique est simulé par un code spécifique, écrit par un spécialiste du domaine. L'application finale de couplage de code assemble les différents programmes comme le montre la figure 2.1 : le programme de contrôle lance l'application et change les paramètres de calcul, le code d'écoulement fluide a été écrit par un physicien numéricien, et le programme de transport réactif a été développé par un chimiste numéricien.

[3] Action Concertée Incitative sur la Globalisation des Ressources Informatiques et des Données, supportée par le Fonds National de la Science du ministère chargé de la recherche.

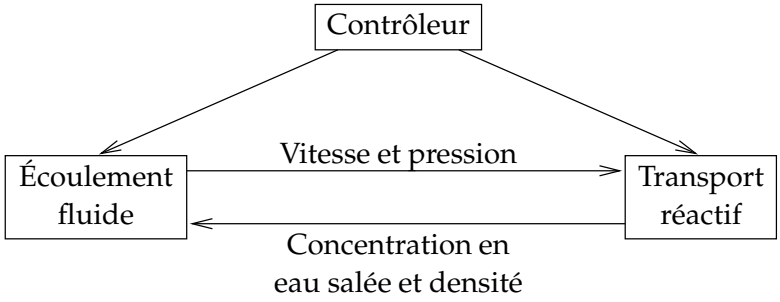


FIG. 2.1 – Schéma d'une application HydroGrid. Chaque rectangle représente un programme.

L'indépendance des composants et la spécification claire de leurs interfaces facilite aussi la maintenance des applications complexes. Il est possible de changer une partie bien localisée d'un programme pour corriger une application, tant que les interfaces sont respectées.

Séparation des codes. La programmation par composants logiciels promeut la séparation du code métier (fonctionnel) et du code de gestion (non-fonctionnel) afin d'accroître la réutilisabilité des composants.

Par exemple, au moment du déploiement d'une application, il peut être nécessaire d'assurer la sécurité des communications entre les composants (chiffrement des données), ou bien la compression et la décompression des données à la volée. Le code de chiffrement ou de compression des données transmises n'appartient pas au code métier (qui reste inchangé), car ces fonctions ne font pas partie de la résolution du problème, et sont activables de manière optionnelle. Ainsi, un chimiste qui écrit un composant pour calculer le point d'équilibre d'une réaction chimique peut se concentrer sur le problème numérique à résoudre sans se soucier de l'environnement dans lequel le composant sera exécuté.

Dynamicité. Les composants peuvent se connecter et déconnecter dynamiquement pendant l'exécution de l'application. Cette propriété permet par exemple de connecter ponctuellement un composant de visualisation à une application de simulation numérique pour observer l'évolution des calculs.

2.2.2.4 Technologies de composants logiciels

Fractal. Les composants Fractal [44, 181] sont développés dans le cadre du consortium ObjectWeb [211]. Leur implémentation de référence est nommée *Julia* [200]. Fractal permet de créer des composants composites (ou hiérarchiques) : de tels composants sont constitués de plusieurs sous-composants qui forment une boîte noire (le composant composite) avec une interface publique. Un composite est flexible : il peut être modifié après instanciation en changeant les relations entre les sous-composants.

En matière de déploiement, Fractal possède un ADL. Un ADL (*Architecture Description Language*) est une description de haut niveau de l'architecture d'une application, c'est-à-dire des parties (ou composants) qui la constituent. L'ADL permet d'exprimer les dépendances des composants, leur assemblage, leur composition hiérarchique ou non, les interconnexions entre

les composants et la configuration de leurs paramètres. L'objectif d'un ADL est de décrire une application sans faire d'hypothèses sur l'environnement dans lequel elle peut être déployée : la création d'un ADL peut donc être considérée comme la toute première étape du déploiement d'une application, mais l'ADL ne suffit pas à déployer *effectivement* l'application. Par exemple, l'ADL ne permet pas de découvrir les ressources d'exécution, ni de placer les composants de l'application sur les ordinateurs.

ICENI. ICENI (the Imperial College e-Science Networked Infrastructure [195, 87]) est un environnement de programmation par composants logiciels. ICENI offre un système intégré pour construire des applications par assemblage de composants, et toute une technologie pour déployer ces applications en environnement distribué. Le projet ICENI se focalise principalement sur l'information concernant le comportement et les performances des composants et de leurs diverses implémentations. Cette information de haut niveau permet ensuite de choisir les meilleures implémentations des composants dont l'application a besoin, en fonction des ressources de calcul disponibles et de l'assemblage de composants.

ProActive. ProActive [219] est une suite logicielle en JAVA qui offre un grand nombre de services : migration d'objets d'une JVM[4] à une autre, sécurité, communications de groupe [29], *etc.* Les composants implémentés dans ProActive suivent le modèle des composants hiérarchiques de Fractal.

En matière de déploiement [33], ProActive élimine du code source les noms de machines, les adresses IP et les protocoles de création de processus, localement ou à distance (via `rsh`, `ssh`, *etc.*). Pour rendre l'application indépendante vis-à-vis des plates-formes d'exécution, ProActive introduit la notion de « nœuds virtuels » (*virtual nodes*). Dans le code source[5], les composants sont instanciés sur des nœuds virtuels. Ensuite, un descripteur (au format XML) associe les nœuds virtuels à des JVM, puis les JVM à des machines (ou adresses IP) et des protocoles de création de JVM (localement ou à distance).

Outre la nécessité de spécifier des noms de nœuds virtuels dans le code source, un inconvénient de cette méthode est que la description d'application et la description de son déploiement ne sont pas indépendantes. En effet, elles sont toutes les deux reliées par les noms de nœuds virtuels. Ainsi, le descripteur de déploiement est statique et spécifique à la fois à une application et à une infrastructure d'exécution : il doit être manuellement regénéré si l'on veut déployer cette même application sur une autre infrastructure.

CCA et XCAT3. CCA (*Common Component Architecture*, [26]) est une technologie de composants logiciels. XCAT3 [110] en est une implémentation pour les environnements distribués. Les applications XCAT3 se déploient sur des Web Services déjà existants, mais CCA ne définit pas de modèle général pour déployer ses applications. Le problème du déploiement est donc reporté de XCAT3 vers les Web Services, qui ne spécifient pas non plus comment ils peuvent être lancés.

[4] JVM : *Java Virtual Machine*, machine virtuelle JAVA.

[5] Le code source devant être modifié, il est impossible d'utiliser des codes propriétaires.

EJB et CCM. Les EJB (*Enterprise Java Beans*, [70]) de Sun Microsystems et CCM (CORBA *Component Model*, [51]) de l'OMG sont des technologies de composants logiciels qui reposent principalement sur l'appel de méthode à distance. Ces technologies de composants présentent l'avantage de spécifier leur modèle de déploiement (qui sont très proches) à travers des modèles de packaging, d'assemblage et de description des applications indépendante des environnements d'exécution.

Discussion. Les technologies de composants les plus matures et qui spécifient un modèle de déploiement sont les EJB et CCM. Les EJB et ProActive sont liés au langage de programmation JAVA. Fractal n'est pas aussi mature que CCM (qui bénéficie d'un support industriel plus important que Fractal). CCM donne lieu à des applications dont la structure est plus simple que celle des applications Fractal, car CCM ne comporte pas la notion de composants composites ni celle de versions des composants (contrairement au versioning de Fractal [43]). Comme CCM est indépendant des langages de programmation et des systèmes d'exploitation, et que son modèle de packaging et déploiement des composants est le plus abouti, nous avons travaillé avec les composants CORBA. En outre, il a déjà été démontré que le calcul haute performance est possible avec CORBA [64] : PadicoTM est une plate-forme de communication pour la programmation des grilles de calcul qui permet de faire communiquer des composants CORBA avec des performances proches de celles du matériel réseau. Par conséquent, les composants fondés sur CORBA sont intéressants pour le calcul scientifique haute performance. Ainsi, nous présentons plus en détails cette technologie de composants logiciels dans la section suivante.

2.2.2.5 Vue d'ensemble du modèle de composants CORBA (CCM)

Le modèle de composants CORBA fait partie des spécifications de CORBA version 3.0 [51] définies par l'OMG. Il représente le premier modèle industriel pour des composants *métier* distribués, hétérogènes, indépendants des langages de programmation, des systèmes d'exploitation et des fournisseurs d'implémentations. CCM spécifie l'intégralité du cycle de vie des composants que nous détaillons ci-dessous : création, exécution, packaging, assemblage, et déploiement.

Création. La production des composants CCM (figure 2.2) consiste à

1. définir leurs interfaces grâce au langage de description d'interface IDL3 (*Interface Definition Language*) : les ports qui *utilisent* une interface sont nommés « réceptacles », et ceux qui *fournissent* une interface sont appelés « facettes » (ces deux types de ports permettent des communications synchrones). La figure 2.2 montre trois autres types de ports CCM : les puits et source d'événements pour les communications asynchrones, et les attributs pour configurer des propriétés du composant ;
2. préciser, facultativement, certains de leurs aspects non-fonctionnels (persistance, transactions) ;
3. puis programmer les composants en écrivant leur code métier (ou code fonctionnel).

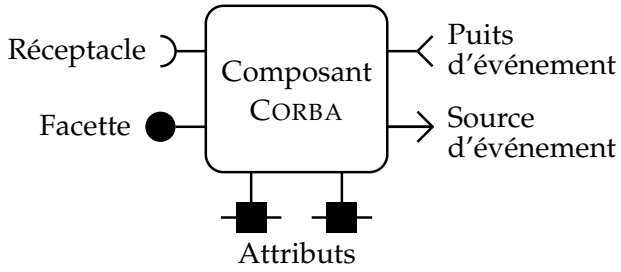


FIG. 2.2 – Différents types de ports d'un composant CCM.

Exécution. CCM définit l'environnement d'exécution des instances de composants. Cet environnement est fondé sur des « conteneurs » qui hébergent les instances de composants, et qui prennent en charges les services non-fonctionnels (persistance, transactions, sécurité, *etc.*). Ainsi, conformément au modèle de composants logiciels, les composants CCM ne contiennent que du code métier, et les aspects non-fonctionnels de l'application sont définis par des descripteurs externes aux composants.

Entre les phases de création des composants et leur exécution, la norme CCM définit des étapes importantes qui permettent le déploiement des composants CORBA : le packaging, l'assemblage, et le déploiement proprement dit.

Packaging. Chaque composant peut être compilé pour différentes plates-formes (systèmes d'exploitation et architectures matérielles). Un composant est décrit par un fichier XML (d'extension `.csd`[6]) qui contient les informations suivantes (figure 2.3) :

- la description des interfaces (ports) du composant, par le biais d'un pointeur vers le fichier IDL3 du composant (élément `idl`) ;
- la description des propriétés du composant (élément `propertyfile`), c'est-à-dire les valeurs initiales de ses attributs (figure 2.2) au moment de l'instanciation du composant (les valeurs des attributs peuvent être modifiées par la suite au cours de l'exécution) ;
- les dépendances du composant, vis-à-vis de classes JAVA, de bibliothèques, d'exécutables, ou tout autre fichier ;
- la liste des implémentations disponibles du composant, c'est-à-dire les versions compilées sous la forme de classe JAVA, d'exécutable, ou de DLL (bibliothèque dynamique, ou « objet partagé » `.so` dans un monde UNIX). Un composant sous forme de DLL devra s'exécuter au sein d'un processus appelé « serveur de conteneur » (*ComponentServer*) et offrir un point d'entrée pour son exécution, défini dans ce fichier de description. Les informations concernant chaque implémentation du composant sont les suivantes :
  - le type d'implémentation (classe JAVA, exécutable ou DLL) ;
  - la localisation du code compilé (URL, fichier local, *etc.*) ;
  - le type de système d'exploitation et d'architecture matérielle pour lesquels l'implémentation a été compilée.

Ensuite, chaque composant peut être « packagé » (ou « conditionné ») en une archive compressée (fichier ZIP d'extension `.zip`). Un package de composant contient :

- obligatoirement le fichier XML `.csd` de description du composant ;

[6] CORBA *Software Descriptor*.

```
<softpkg name="Bank" version="1,0,1,0">

   <pkgtype>CORBA Component</pkgtype>
   <title>Bank</title>
   <description>Yet another bank example</description>

   <!-- description des ports du composant -->
   <idl id="IDL:M1/Bank:1.0">
      <link href="ftp://x/y/Bank.idl">
   </idl>

   <!-- description des propriétés du composant -->
   <propertyfile>
      <fileinarchive name="bankprops.cpf"/>
   </propertyfile>

   <!-- première implémentation disponible -->
   <implementation id="DCE:700dc518-0110-11ce-ac8f-0800090b5d3e">
      <!-- plate-forme d'exécution de cette version du composant -->
      <os name="WinNT" version="4,0,0,0"/>
      <os name="Win95"/>
      <processor name="x86"/>
      <compiler name="MyFavoriteCompiler"/>
      <programminglanguage name="C++"/>
      <!-- le fichier compilé du composant sous forme de DLL -->
      <code type="DLL">
         <fileinarchive name="bank.dll"/>
         <entrypoint>createBankHome</entrypoint>
      </code>
      <!-- deux dépendances -->
      <dependency type="ORB">
         <name>ThisORB</name>
      </dependency>
      <dependency type="DLL">
         <localfile name="rwthr.dll"/>
      </dependency>
   </implementation>

   <!-- deuxième implémentation disponible -->
   <implementation id="DCE:297f3e18-0110-11ce-ac8f-08074982ad3e">
      <os name="Solaris" version="5,5,0,0"/>
      <processor name="sparc"/>
      <code type="Executable">
         <!-- URL de l'exécutable de cette version du composant -->
         <link href="http://exec.store.com/sparc/bank.exe"/>
      </code>
   </implementation>

</softpkg>
```

FIG. 2.3 – Exemple de fichier XML **`.csd`** de description d'un composant.

– éventuellement les versions compilées du composant pour une ou plusieurs plates-formes ; si une implémentation du composant ne se trouve pas dans l'archive compressée, alors elle est localisée à une URL spécifiée par le fichier XML de description du composant.

Assemblage. Afin de construire une application CCM, plusieurs composants peuvent être « assemblés » (ou composés) en interconnectant leurs différents ports (dans le respect des types d'interfaces). Un assemblage de composants est décrit par un fichier XML (d'extension **`.cad`**[7]) qui contient les informations suivantes (figure 2.4) :

– les descriptions de chaque composant (`componentfile`) qui constitue l'assemblage, par le biais de pointeurs (URL, fichier local, *etc.*) vers les descripteurs XML **`.csd`** des composants ;
– dans la section `partitioning`, les instanciations de composants (`homeplacement`), avec des cardinalités qui peuvent être spécifiées, ainsi que des paramètres particuliers de configuration des attributs pour chaque instance de composant ;
– les connexions entre les instances de composants, qui définissent quels ports de quelles instances de composants devront être interconnectés lors de la phase de déploiement ;
– les contraintes de placement des instances de composants, qui peuvent se décliner sous trois formes :
  • aucune contrainte sur le placement d'une instance de composant ;
  • `processcollocation` désigne une contrainte de « co-localisation dans un même processus », qui impose que plusieurs instances de composants s'exécutent au sein d'un même processus (le serveur de conteneur, ou *ComponentServer*). Ce cas de figure n'a de sens que pour les composants sous la forme de DLL : il permet par exemple à plusieurs instances de composants de partager un même espace d'adressage virtuel ;
  • et `hostcollocation` indique une contrainte de « co-localisation sur une même machine », qui impose que plusieurs instances de composants s'exécutent sur un même ordinateur. Cette contrainte permet par exemple à plusieurs instances de composants de partager un espace de mémoire physique commun, ou bien des fichiers sur un même système de fichiers.

Ensuite, tout assemblage de composants (*i.e.*, toute application CCM) peut être packagé en une archive compressée (fichier ZIP d'extension **`.aar`**[8]). Un package d'assemblage de composants contient :

– obligatoirement le fichier XML **`.cad`** de description de l'assemblage de composants ;
– éventuellement les fichiers **`.csd`** de description des composants individuels qui constituent l'application, sinon le descripteur **`.cad`** de l'assemblage contient des pointeurs (URL) vers ces fichiers **`.csd`** ;
– éventuellement les implémentations des composants qui forment l'application, sinon le descripteur **`.cad`** de l'assemblage contient des pointeurs (URL) vers ces versions compilées des composants.

Déploiement. CCM prévoit le déploiement de composants individuels (**`.csd`** ou **`.zip`**) ainsi que le déploiement d'assemblages de composants (**`.cad`** ou **`.aar`**). Dans son modèle de dé-

[7] *Component Assembly Descriptor*.
[8] *Assembly ARchive*.

```
<componentassembly id="ZZZ123">
   <componentfiles>               <!-- liste des composants de l'assemblage -->
      <componentfile id="composant_A">
         <fileinarchive name="comp_a.csd"/>
      </componentfile>
      <componentfile id="composant_B">
         <link href="ftp://www.xyz.com/comp_b.csd"/>
      </componentfile>
   </componentfiles>

   <partitioning>            <!-- instanciation des composants et placement -->
      <homeplacement id="instance_A0">           <!-- pas de contrainte ici -->
         <componentfileref idref="composant_A"/>
         <homeproperties>
            <fileinarchive name="AHomeProperties.cpf"/>
         </homeproperties>
      </homeplacement>
      <processcollocation cardinality="3"> <!-- 2 instances dans 1 processus -->
         <homeplacement id="instance_A1">    <!-- l'ensemble répliqué 3 fois -->
            <componentfileref idref="composant_A"/>
         </homeplacement>
         <homeplacement id="instance_B1">
            <componentfileref idref="composant_B"/>
         </homeplacement>
      </processcollocation>
      <hostcollocation>   <!-- 3 instances de composants sur la même machine -->
         <processcollocation>            <!-- dont 2 dans le même processus -->
            <homeplacement id="instance_A2.0">
               <componentfileref idref="composant_A"/>
            </homeplacement>
            <homeplacement id="instance_B2.0">
               <componentfileref idref="composant_B"/>
            </homeplacement>
         </processcollocation>
         <homeplacement id="instance_B2.1">
            <componentfileref idref="composant_B"/>
         </homeplacement>
      </hostcollocation>
   </partitioning>

   <connections>                 <!-- listes des connexions entre les ports -->
      <connectinterface>
         <usesport>                                    <!-- un réceptacle... -->
            <usesidentifier>interface_ABC</usesidentifier>
            <componentinstantiationref idref="instance_A1"/>
         </usesport>
         <providesport>                 <!-- ... et sa facette du même type -->
            <providesidentifier>interface_ABC</providesidentifier>
            <componentinstantiationref idref="instance_B2.0"/>
         </providesport>
      </connectinterface>
      <!-- et d'autres connexions ici... -->
   </connections>

</componentassembly>
```

FIG. 2.4 – Exemple de fichier XML **`.cad`** de description d'assemblage de composants.

ploiement, CCM spécifie les étapes suivantes :

1. l'installation des implémentations des composants sur chaque plate-forme d'exécution de l'application ;
2. l'instanciation de chaque composant, qui consiste à lancer les exécutables des composants, ou bien les serveurs de conteneur qui chargent les DLL des composants (en respectant les éventuelles contraintes de co-localisation) ;
3. la configuration des instances de composants, en fixant les valeurs initiales de leurs attributs ;
4. l'interconnexion des instances de composants conformément à la description `.cad` dans le cas d'un assemblage de composants ;
5. et enfin l'activation de chaque instance de composant, qui lance l'exécution de l'application.

Le modèle de déploiement de CCM est dynamique : les composants peuvent être connectés et déconnectés au cours de l'exécution de l'application, et des instances de composants peuvent être ajoutées ou supprimées dynamiquement. Les fichiers XML `.cad` de description des assemblages de composants définissent des interconnexions entre les instances de composants, mais il s'agit uniquement de l'*état initial* de l'application à son lancement.

2.2.3 Discussion

La programmation par composants logiciels est une technologie intéressante pour le calcul scientifique : sa modularité permet de construire des applications complexes de couplage de codes faisant coopérer plusieurs programmes de simulation numérique. Dans le cadre de cette technologie, la norme CCM des composants CORBA offre le modèle le plus abouti, en particulier pour ce qui concerne le déploiement des applications.

Cependant, même si le modèle de déploiement de CCM est le plus abouti parmi les diverses technologies de composants logiciels, il reste des points que CCM ne spécifie pas :

- comment sont sélectionnées les machines sur lesquelles s'exécutent les composants (*i.e.*, le « placement » des instances de composants) ;
- comment sont transférées les implémentations des composants (classes JAVA, exécutables, DLL) de leurs points de stockage vers les machines sélectionnées pour exécuter l'application dans un environnement éventuellement distribué ;
- comment sont lancés les processus sur les machines distantes.

CCM ne peut pas définir ces points, car ils sont trop dépendants de l'environnement d'exécution choisi par l'utilisateur qui veut déployer son application. Or il se trouve que CCM veut rester indépendant des infrastructures d'exécution.

2.3 Applications parallèles

Les applications parallèles sont une classe très importante d'applications concurrentes pour le calcul scientifique : elles visent à diminuer les temps d'exécution des calculs numériques intensifs, ou bien à répartir les données pour résoudre des problèmes de grande taille qui satureraient la mémoire d'un seul ordinateur ou même d'un seul super-calculateur. Cette section présente la classe des applications parallèles, puis elle se focalise sur deux technologies

de programmation parallèle sur lesquelles nous avons travaillé, à savoir les bibliothèques MPI et les systèmes de MVP.

2.3.1 Programmation parallèle

2.3.1.1 Objectifs de la programmation parallèle

La programmation parallèle s'est imposée dans le domaine du calcul scientifique, car son objectif est essentiellement la haute performance. La programmation parallèle vise :

- à répartir les tâches de calcul dans plusieurs processus sur plusieurs processeurs ;
- à répartir les données des problèmes de grande taille qui satureraient la mémoire d'un seul ordinateur ou même d'un super-ordinateur[9] ;
- à recouvrir les calculs et les opérations d'entrées-sorties, afin de masquer leur latence.

Dans tous les cas, l'objectif de la programmation parallèle est de diminuer les temps d'exécution des calculs numériques intensifs. Cette propriété permet alors d'augmenter la précision des calculs (et donc la taille des données) tout en conservant des temps de calcul acceptables.

2.3.1.2 Panorama de la programmation parallèle

Programmation multi-threads ou multi-processus. La programmation « multi-threads » est une méthode de programmation parallèle qui consiste à faire coopérer plusieurs threads au sein d'un même processus pour exécuter l'application : elle permet en particulier de recouvrir les calculs et les opérations d'entrées-sorties. Une autre approche est la programmation « multi-processus »[10] : soit tous les processus de l'application sont issus du même code source (SPMD[11]), soit l'application est faite de plusieurs programmes (ou codes source) différents (MPMD[12]). Dans le cas de la programmation multi-processus, les tâches de calcul sont le plus souvent réparties sur des ordinateurs qui ne partagent pas de mémoire commune. Or les processus doivent coopérer, donc s'échanger des données ou se synchroniser. Les communications entre les processus sont susceptibles de ralentir l'exécution d'une application parallèle multi-processus. Le ratio calcul/communications est le rapport entre le temps passé par les processus à faire des opérations de calcul et le temps qu'ils passent à communiquer. Plus ce ratio est faible et moins l'intérêt de la parallélisation est évident. C'est pourquoi on peut rarement augmenter indéfiniment le nombre de processus qui se répartissent les tâches de calcul, car ils passent alors plus de temps à communiquer qu'à calculer, et le temps global d'exécution de l'application augmente.

Applications statiques et applications dynamiques. Parmi les applications parallèles, on peut distinguer les applications statiques et les applications dynamiques.

[9] Lorsque la mémoire physique est saturée, les systèmes d'exploitation modernes copient des pages de mémoire dans la zone de « *swap* » du disque dur ; comme les temps d'accès aux disques sont nettement supérieurs aux temps d'accès aux mémoires, les performances d'exécution sont considérablement réduites.

[10] Les approches multi-threads et multi-processus ne sont pas incompatibles.

[11] *Single Program, Multiple Data*.

[12] *Multiple Programs, Multiple Data*.

Applications statiques. Dans une application statique, tous les processus sont lancés au moment du déploiement de l'application : l'utilisateur sait combien de processus composent l'application au début de son exécution, et le nombre de processus reste le même tout au long de l'exécution.

Applications dynamiques. Dans une application dynamique, il se peut qu'un ou plusieurs processus soient créés en cours d'exécution, ou bien se terminer avant la fin de l'exécution de l'application. Cette création ou terminaison de processus en cours d'exécution se fait *à l'initiative* de l'application, et les processus ajoutés doivent faire partie intégrante de l'application, au même titre que les processus lancés au moment du déploiement initial de l'application. Par exemple, un utilisateur qui lance par ailleurs un programme de visualisation ou d'analyse des traces de l'application en cours d'exécution ne permet pas de qualifier l'application de dynamique.

Actuellement, la plupart des applications de calcul scientifique sont statiques, car elles sont plus simples à programmer, et il existe peu de support pour les applications parallèles dynamiques.

2.3.1.3 Modèles de programmation parallèle

Nous distinguons deux grandes classes de modèles de programmation parallèle :
- avec le parallélisme *explicite*, le programmeur fait l'effort de parallélisation de l'application : décomposition en sous-tâches, placement des tâches sur les processeurs, répartition et redistribution des données, communications (échange de données et synchronisation) entre les tâches, *etc.* ;
- avec le parallélisme *implicite*, le travail de parallélisation de l'application est effectué par un compilateur (pour les langages parallèles), par un outil de parallélisation automatique, ou bien par une bibliothèque de fonctions déjà parallélisées.

Comme le programmeur est souvent le plus à même de décider comment le parallélisme peut être exploité pour un problème particulier, le parallélisme explicite donne en général de meilleures performances que la parallélisme implicite.

La programmation par passage de messages, celle par mémoire partagée et certains langages parallèles font partie de la classe du parallélisme explicite ; les bibliothèques parallèles de haut niveau d'abstraction relèvent du parallélisme implicite, même si la frontière entre ces deux types de parallélisme n'est pas toujours très nette.

Passage de messages. La programmation par passage de messages consiste à faire coopérer les différentes tâches qui constituent l'application par l'*envoi* et la *réception explicites* de données, typées ou non. Le plus souvent, les environnements de programmation par passage de messages offrent également des opérations de synchronisation simples (la « barrière », qu'un processus ne peut franchir que lorsque tous les autres processus de l'application l'ont aussi atteinte), ainsi que des opérations « collectives », telles que :
- envoyer une donnée à tout un groupe de processus en une seule opération (diffusion, ou *Broadcast*) ;
- effectuer une opération arithmétique simple (produit, somme, *etc.*) ou logique (« et », « ou », *etc.*) sur des données détenues par tous les processus d'un groupe (opération de « réduction », *Reduce*) ;

– regrouper dans un seul processus des données qui sont dispersées dans chaque processus d'un groupe (*Gather*) ;
– disperser des données situées dans un vecteur sur un processus vers les autres processus d'un groupe (*Scatter*) ;
– *etc.*

Ainsi, la programmation par passage de messages est un modèle d'assez bas niveau : le travail de parallélisation revient intégralement au programmeur.

Mémoire partagée. Dans le modèle de programmation parallèle par mémoire partagée, les tâches qui constituent l'application coopèrent par l'*écriture* et la *lecture* dans une *mémoire commune* : ces opérations permettent d'échanger des données et de se synchroniser.

Si les tâches sont les threads d'un même processus, alors ils peuvent lire et écrire dans l'espace d'adressage du processus. Si les tâches sont des processus au sein d'un même système d'exploitation, alors ils peuvent partager des segments de mémoire : c'est le cas par exemple avec les IPC[13], ou bien encore avec les systèmes à image unique (SSI[14]). Enfin, le mécanisme de « mémoire virtuellement partagée » (MVP) permet de programmer des systèmes à mémoire distribuée en utilisant le modèle de programmation parallèle par mémoire partagée : un environnement de MVP donne l'illusion à des processus localisés sur des machines distribuées (avec des systèmes d'exploitation distincts) de partager un espace mémoire commun.

Dans ce modèle, la distribution des données est transparente et les communications entre les tâches de l'application parallèle sont implicites : ce modèle est d'un peu plus haut niveau que celui de la programmation par passage de messages. Cependant, la programmation par mémoire partagée laisse encore le soin au programmeur de gérer lui-même la décomposition du problème en tâches de calcul, le placement de ces tâches, et la gestion de la localité des accès.

Langages parallèles. Avec les langages parallèles, c'est le compilateur et/ou l'environnement d'exécution qui gèrent le parallélisme : le programmeur se contente de donner des indications d'ordre sémantique (ou directives) pour aider le compilateur à exploiter le parallélisme le mieux possible.

HPF (*High Performance Fortran*, [95]) est un langage à *parallélisme de données*, pour réaliser simplement une même opération sur un ensemble de données. HPF est fondé sur le langage FORTRAN, mais il exige une structure régulière d'organisation des données, et il est relativement peu utilisé : le site web du Forum HPF [192] répertorie 35 applications HPF.

OPENMP est une norme qui définit un ensemble de directives valables pour plusieurs langages de programmation (C, C++, FORTRAN), et qui permettent d'exposer du *parallélisme de contrôle* (sur les boucles, *etc.*). OPENMP repose sur l'utilisation de threads qui communiquent par mémoire partagée.

D'autres langages parallèles existent, tels que SISAL (*Streams and Iteration in a Single Assignment Language*, [71]) ou PCN (*Program Composition Notation*, [82]), mais ils n'ont jamais vraiment réussi à s'imposer dans la communauté du calcul scientifique.

[13] *Inter-Process Communication*.
[14] *Single System Image*, où un système d'exploitation unique couvre un ensemble d'ordinateurs distribués, tel que Kerrighed : `http://www.kerrighed.org/`.

Bibliothèques parallèles de haut niveau d'abstraction. Compte tenu de la complexité de la programmation par passage de messages ou par mémoire partagée, des bibliothèques de plus haut niveau d'abstraction sont proposées pour résoudre des tâches classiques du calcul parallèle. Une seule abstraction (recherche des valeurs propres d'une matrice, diagonalisation, produit, *etc.*) se fait en parallèle en encapsulant les primitives de contrôle, de synchronisation et d'échanges de données entre les tâches qui s'exécutent en parallèle.

L'un des avantages des bibliothèques de haut niveau d'abstraction est de faciliter la programmation parallèle, en rendant la parallélisation implicite, ce qui permet au programmeur de se concentrer sur le problème scientifique à résoudre, plutôt que sur les aspects techniques de bas niveau d'utilisation des environnements de programmation parallèle. Ces bibliothèques spécialisées de haut niveau d'abstraction peuvent être implémentées sur des environnements de programmation par passage de messages, ou bien au dessus d'un système de mémoire partagée (par exemple un système de MVP sur des machines distribuées).

2.3.1.4 Discussion

Cette section a présenté un échantillon significatif des modèles de programmation parallèle utilisés dans le domaine du calcul scientifique. D'une part, la parallélisation explicite est massivement utilisée au travers des bibliothèques de passage de messages et des environnements de mémoire partagée. D'autre part, la parallélisation implicite, par le biais de bibliothèques de haut niveau d'abstraction et de certains langages parallèles, s'appuie également sur le passage de messages ou la mémoire partagée. Ainsi, les deux sections suivantes présentent plus en détails ces deux paradigmes importants de programmation parallèle qui sont, directement ou non, à l'origine de la plupart des applications parallèles de calcul scientifique.

2.3.2 Programmation parallèle par passage de messages

Comme l'a montré la section précédente, le modèle de programmation parallèle par passage de messages est très largement utilisé dans le domaine du calcul scientifique, que ce soit directement par le programmeur, ou bien indirectement via des bibliothèques de haut niveau. Les bibliothèques de passage de messages permettent d'écrire des applications parallèles en général performantes pour des systèmes à mémoire distribuée.

2.3.2.1 MPI et PVM

Les deux environnements de programmation par passage de messages les plus utilisés sont les bibliothèques MPI (*Message-Passing Interface*, [91, 148]) et PVM (*Parallel Virtual Machine*, [88]). MPI est une API[15] pour écrire des programmes parallèles portables. PVM est une bibliothèque portable de programmation parallèle sur des ordinateurs distribués, hétérogènes (systèmes d'exploitation, architectures matérielles), et reliés en réseau. PVM est plus ancien, mais nous avons choisi de travailler avec MPI qui présente de nombreux avantages :

– MPI est une *norme* définie par le « MPI Forum » [206], soutenu par les milieux académiques et industriels ;

[15] *Application Programming Interface*.

– il existe un très grand nombre d'implémentations de MPI, tant académiques (gratuites ou non) qu'industrielles (IBM, Sun Microsystems, SGI, *etc.*), et qui sont portées sur de nombreuses plates-formes ;
– MPI est toujours un sujet de recherche actif et fait l'objet de nombreux développements ;
– MPI est très massivement utilisé dans les milieux scientifiques (physiciens, chimistes, *etc.*) pour le calcul numérique parallèle.

Le plus souvent, une application MPI est constituée d'un seul programme (SPMD) qui s'exécute sur des machines à mémoires distribuées, éventuellement compilé pour des systèmes d'exploitation et des architectures matérielles différents. Chaque processus de l'application peut établir un canal de communication directement avec chacun des autres processus de l'application MPI. Les processus MPI peuvent se regrouper au sein de « communicateurs », qui sont des groupes de processus créés dynamiquement en cours d'exécution. À l'intérieur d'un communicateur, MPI spécifie un ensemble d'opérations collectives (*broadcast*, *reduce*, *gather*, *scatter*, *barrier*, *etc.*).

Le degré de parallélisme d'une application MPI est le nombre de processus MPI qui font partie de l'application. Le nombre de processus MPI est déterminé au moment du lancement de l'application. Depuis la version 2 de la norme MPI, il est possible d'écrire des applications MPI dynamiques : par exemple, la fonction `MPI_Comm_spawn` permet à l'application d'ajouter des processus MPI à ceux lancés initialement.

Pour ce qui est du déploiement d'applications MPI, la norme n'impose aucune contrainte sur les implémentations de MPI afin de ne pas restreindre son champ d'utilisation. Elle se contente de *suggérer* une commande `mpiexec` dont elle définit la syntaxe, et qui permet à l'utilisateur de spécifier les implémentations de l'application (et le nombre de leurs instances) qui devront s'exécuter sur une liste de machines. L'utilisateur a la responsabilité de faire en sorte que les bonnes implémentations soient installées sur chaque machine.

2.3.2.2 Implémentations de MPI

Il existe un très grand nombre d'implémentations de MPI, chacune avec ses caractéristiques :

– MPICH[16], LAM/MPI[17] sont des implémentations génériques haute performance ;
– LA-MPI[18], FT-MPI[19] tolèrent les défaillances (des réseaux pour LA-MPI, ou des processeurs pour FT-MPI) ;
– PACX-MPI[20] [104, 215], MagPIe[21] [107], MPICH-G2[22] [101, 205] sont adaptées aux fédérations de clusters et aux grilles de calcul (voir le chapitre 3) ;
– OpenMPI vise à être « la meilleure implémentation » haute performance de la norme MPI, et s'inspire de FT-MPI, LA-MPI, LAM/MPI et PACX-MPI ;
– IBM, Sun Microsystems, SGI proposent leurs implémentations de MPI, spécialisées pour les ordinateurs et systèmes d'exploitation qu'ils commercialisent.

[16] MPICH-1 et MPICH-2, d'Argonne National Laboratory, USA.
[17] Indiana University, USA.
[18] Los Alamos MPI.
[19] Fault-Tolerant MPI, University of Tennessee, USA.
[20] HLRS, Stuttgart, Allemagne.
[21] Vrije Universiteit, Amsterdam, Pays-Bas.
[22] Argonne National Laboratory, USA.

Ces implémentations de MPI ne sont pas intéropérables, et chacune choisit ses mécanismes de lancement de processus à distance. Par exemple, MPICH-1 a besoin d'un fichier de configuration statique qui liste les machines sur lesquelles peuvent être lancés les processus MPI : tous les processus sont lancés à distance via la même méthode de soumission (`ssh` ou `rsh`). LAM/MPI et MPICH-2 supposent que des daemons s'exécutent sur les ressources de calcul disponibles : ces daemons sont chargés de lancer les processus MPI sur les machines distribuées. Ces implémentations de MPI fournissent des outils sommaires qui permettent de lancer les daemons, à la manière de MPICH-1, avec une liste statique de machines et un unique protocole de lancement de processus à distance.

2.3.2.3 Discussion

La bibliothèque de programmation parallèle MPI a beaucoup de succès dans le domaine du calcul scientifique. Cependant, il n'existe pas de modèle de déploiement pour les applications MPI, ni même de consensus sur la méthode à employer pour les lancer. Les outils de déploiement des applications MPI sont très sommaires et présentent d'importantes limitations, car MPI a longtemps été utilisé pour des applications de structure peu complexe et sur des infrastructures d'exécution relativement simples (machines parallèles ou clusters homogènes) : la liste des machines d'exécution doit être explicitement spécifiée, les protocoles de soumission de tâches sont assez restreints, l'utilisateur est responsable de faire en sorte que les bonnes implémentations de l'application soient effectivement installées sur chaque machine, en fonction de leurs systèmes d'exploitation et de leurs architectures matérielles.

Même si elle est massivement utilisée, la technique de programmation par passage de messages est assez contraignante, car c'est le programmeur qui doit gérer explicitement la répartition des données dans les mémoires distribuées des processus, ainsi que la redistribution des données au cours de l'exécution de l'application parallèle. De plus, la communication de structures de données complexes (avec des pointeurs, par exemple) est très délicate avec MPI par exemple. Le modèle de programmation par mémoire partagée, présenté dans la section suivante, évite ces difficultés au programmeur.

2.3.3 Programmation parallèle par mémoire partagée

Le modèle de programmation par mémoire partagée est aussi un paradigme majeur du calcul scientifique, que ce soit pour une application directement écrite suivant ce modèle, ou bien comme support du langage OpenMP, ou encore dans l'implémentation de bibliothèques de haut niveau d'abstraction.

2.3.3.1 Mémoire virtuellement partagée

Dans les cas où la mémoire est partagée entre les threads d'un même processus ou entre les processus d'une même machine, le partage de mémoire est réalisé par le système d'exploitation (espace d'adressage des processus, IPC, SSI) ou par des mécanismes matériels (pour les SMP[23] par exemple, figure 2.5). En revanche, pour exécuter des applications parallèles à

[23] *Symmetric Multi-Processors*, ou multi-processeurs à mémoire partagée, qui rassemblent sur un même bus système plusieurs processeurs et un banc mémoire auquel ils accèdent tous de la même manière.

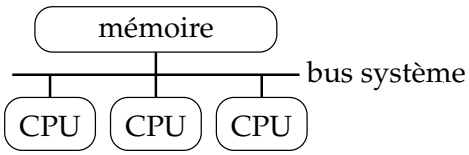


FIG. 2.5 – Multi-processeur à mémoire partagée.

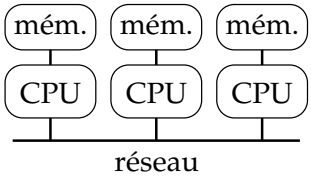


FIG. 2.6 – Système à mémoire distribuée.

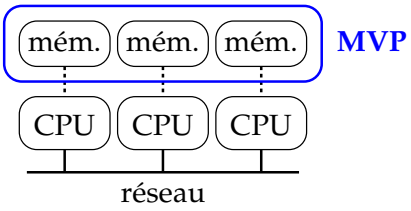


FIG. 2.7 – Architecture à mémoire virtuellement partagée.

mémoire partagée sur des machines distribuées, un système de MVP est nécessaire, car les processeurs possèdent leurs propres bancs mémoire et ils sont reliés par un réseau de communication générique (*clusters*, ou « grappes d'ordinateurs », figure 2.6).

En 1986, Kai Li a présenté le concept de *mémoire virtuellement partagée* (MVP[24], [114, 115]). Dès 1992, ce concept était porté sur machines parallèles [112]. La MVP est un espace mémoire qui est *physiquement distribué*, mais qui apparaît *virtuellement partagé* au programmeur (figure 2.7). À l'instar d'un espace de mémoire habituel (mémoire physique ou espace d'adressage d'un processus), un espace de MVP est linéaire et s'accède via des opérations de lecture et écriture. Évidemment, en environnement concurrent, les accès en lecture ou écriture doivent avoir lieu au sein de sections critiques, délimitées par des opérations de synchronisation ou d'exclusion mutuelle.

Un système de MVP se charge de localiser les données de façon transparente, et de les rapatrier dans le processus qui veut y accéder. Dans la plupart des systèmes de MVP, certaines données partagées peuvent être *répliquées* dans des « caches » (mémoires locales) pour augmenter le parallélisme et améliorer les performances des applications. Schématiquement, le rôle d'un système de MVP peut se décomposer en trois actions :

- projeter des régions de mémoire physique distinctes dans une seule et même mémoire virtuelle ;
- localiser et rapatrier les données pour y accéder ;
- préserver la cohérence de l'ensemble de l'espace mémoire tel qu'il est « vu » par chacun des nœuds de calcul, puisque certaines données peuvent être répliquées.

Le *modèle* de cohérence d'un système de MVP spécifie l'ordre dans lequel un processus « voit » les accès à la mémoire faits par les autres processus, soit sur le même nœud, soit sur un autre nœud : la mémoire est cohérente si la valeur retournée par une opération de lecture correspond toujours à la *dernière* valeur écrite. Tout le problème est de définir la notion de « *dernière* valeur écrite » pour des exécutions parallèles asynchrones sur des processeurs indépendants, sans horloge globale. Ainsi, le modèle de cohérence mémoire est un « contrat » entre le système de MVP et le programmeur de l'application parallèle. Ce contrat précise, lors de la

[24] DSM, *Distributed Shared Memory*.

modification d'une variable partagée, quels processus seront informés de cette modification, et à quel moment.

Le *protocole* de cohérence est une implémentation particulière d'un modèle de cohérence mémoire : à un modèle donné peuvent correspondre plusieurs protocoles, plus ou moins élaborés. En effet, l'implémentation d'un protocole de cohérence mémoire nécessite de faire différents choix :

- le niveau d'implémentation (matériel ou logiciel) ;
- la granularité du partage, c'est-à-dire la taille unitaire des blocs de données manipulés par le système de MVP (les MVP logicielles reposent généralement sur l'abstraction de mémoire virtuelle proposée par le système d'exploitation et sur le mécanisme de défaut de page, donc la granularité du partage de ces MVP est souvent la page mémoire, soit 4 kB ou 8 kB) ;
- les stratégies algorithmiques de maintien de la cohérence mémoire (réplication ou migration de page, lecteurs ou écrivains multiples, *etc.*).

La mise en œuvre d'un système de MVP nécessite de faire plusieurs choix, qui conditionnent à la fois sa simplicité d'utilisation, ses performances et la complexité de sa réalisation. La section suivante détaille l'implémentation de MVP avec laquelle nous avons travaillé : DSM-PM2 présente les avantages de performance et de souplesse d'utilisation que nous présentons dans la suite.

2.3.3.2 DSM-PM2

DSM-PM2 [22] est un système de MVP logicielle de niveau utilisateur qui permet d'implémenter des protocoles de cohérence en environnement multi-threads. Ce système offre une « boîte à outils » générique pour expérimenter et comparer équitablement et assez facilement de nouveaux protocoles de cohérence mémoire, sans avoir à reprogrammer toute l'architecture initiale commune aux MVP : la plate-forme fournit les primitives de synchronisation utiles à la programmation multi-threads. DSM-PM2 est intégré à la suite logicielle PM2 (*Parallel Multithreaded Machine*, [128]) : c'est un environnement multi-threads qui repose sur la bibliothèque de threads nommée « Marcel »[25] et sur la bibliothèque de communication « Madeleine3 »[26] [28]. PM2 met en œuvre le mécanisme d'appel de procédure à distance (RPC) pour faire communiquer les processus.

DSM-PM2 possède ses propres mécanismes de lancement d'application *ad hoc* : l'utilisateur doit spécifier la liste statique des machines d'exécution et le nombre d'instances de processus à lancer. Comme pour MPI, les outils de déploiement sont sommaires, et le choix des protocoles de lancement de processus à distance est limité. L'utilisateur est responsable de faire en sorte que les bonnes implémentations de l'application soient effectivement installées sur chaque machine, en fonction de leurs systèmes d'exploitation et de leurs architectures matérielles.

[25] Bibliothèque de threads fondée sur la norme POSIX [136] et qui gère des threads de *niveau utilisateur*, ce qui lui confère de bonnes performances. En effet, cette bibliothèque fait l'économie des appels système coûteux en temps pour la création, la destruction et l'ordonnancement des threads.

[26] Bibliothèque de communication orientée haute performance et qui s'adapte à différents protocoles de communication sur divers matériels réseaux : SISCI sur SCI [146], TCP sur Ethernet, BIP [137] sur Myrinet [38], *etc.*

2.3.4 Discussion

La programmation parallèle s'est largement imposée dans le domaine du calcul scientifique, par le biais notamment de la programmation par passage de messages et de la programmation par mémoire partagée. Que ces deux paradigmes soient utilisés directement par le programmeur qui écrit l'application, ou bien au travers de langages parallèles et de bibliothèques de haut niveau d'abstraction, ils permettent d'atteindre l'objectif de haute performance du calcul scientifique :

- PETSc[27] [218] est une bibliothèque parallèle de résolution numérique d'équations aux dérivées partielles qui repose sur MPI et qui fait état d'environs 200 applications ;
- ScaLAPACK [226] est une bibliothèque parallèle d'algèbre linéaire (diagonalisation de matrices, résolution de systèmes linéaires, *etc.*) qui fonctionne sur MPI ou PVM ;
- FFTW (*Fastest Fourier Transform in the West*, [180]) est une bibliothèque de transformation de Fourier discrète qui peut reposer soit sur la mémoire partagée (programmation multithreads), soit sur le passage de messages avec MPI.

Deux représentants importants de ces deux modèles de programmation parallèle sont MPI et les systèmes de MVP, qui permettent à l'application parallèle de s'exécuter dans un environnement distribué et hétérogène. Cependant, ces modèles de programmation ne comprennent pas de modèle de déploiement : MPI suggère (sans imposer) une commande de lancement `mpiexec` minimaliste, et chaque implémentation de MVP possède son propre mécanisme *ad hoc* de déploiement rudimentaire. En effet, ces modèles de programmation ont vu le jour à une époque où l'architecte qui concevait les applications, le développeur qui écrivait le code source et l'utilisateur qui déployait ces applications n'était en fait qu'un seul et même individu.

De nos jours, pour obtenir des simulations numériques toujours plus réalistes, les algorithmes sont devenus plus spécialisés, les environnements d'exécution plus sophistiqués (voir le chapitre 3), et donc les applications parallèles sont devenues plus complexes. Pour faire face à cette complexité croissante, la section suivante présente une dernière classe d'applications, qui permet de mieux structurer les applications faites de plusieurs programmes parallèles.

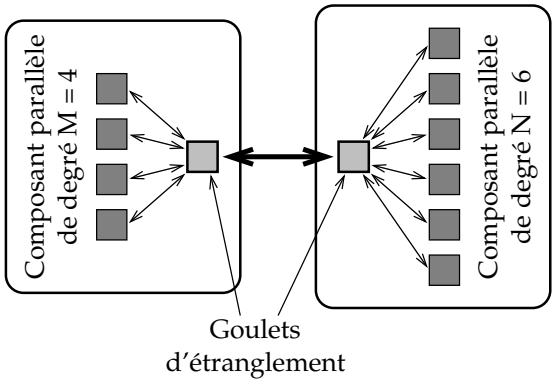
2.4 Applications mixtes

La section 2.2 a mis en exergue quelques propriétés intéressantes des composants logiciels pour les applications de calcul scientifique : connexion dynamique de programmes, support de l'hétérogénéité, structuration des applications, *etc.* La section 2.3, quant à elle, a souligné les propriétés de performance des applications parallèles. Cette section présente un modèle récent de programmation des applications de calcul scientifique, qui combine les modèles de programmation parallèle et de programmation distribuée par composants logiciels.

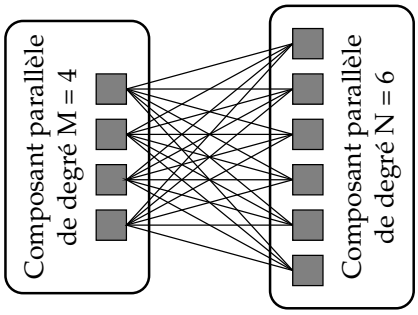
2.4.1 Motivations

La plupart des codes scientifiques actuels sont parallèles. Or la facilité de programmation avec les composants logiciels et l'intérêt de la réutilisation du code incitent de plus en plus

[27] *Portable, Extensible Toolkit for Scientific Computation*, développé à Argonne National Laboratory, USA.



(a) Goulet d'étranglement dans les communications entre deux composants parallèles.



(b) Redistribution M×N des données entre deux composants parallèles.

FIG. 2.8 – Communications entre deux composants parallèles.

à recourir à ce modèle de programmation, d'autant plus que les applications de calcul scientifique deviennent de plus en plus complexes à construire et à gérer. C'est le cas notamment des applications de couplage de codes qui visent des simulations numériques toujours plus réalistes et plus précises.

Une application de couplage de codes fait coopérer plusieurs programmes qui simulent différentes lois physiques, chimiques, biologiques, *etc.* Par exemple, pour concevoir un satellite, on peut vouloir simuler son comportement dans l'espace : cette simulation numérique fait intervenir des codes de thermodynamique, de mécanique du solide, d'optique, de cinétique, *etc.* (applications dites « multi-physiques »). Chaque code (ou programme) est spécialisé, et écrit par un expert du domaine. Il peut être parallèle pour plus de performance.

Les applications mixtes combinent la programmation distribuée par composants logiciels et la programmation parallèle : elles permettent de bénéficier à la fois des performances du parallélisme ainsi que de l'aide à la structuration des applications, de la dynamicité et du support de l'hétérogénéité apportés par la programmation par composants logiciels. En particulier, les applications à base de composants parallèles sont bien adaptées pour le couplage de codes. La section suivante présente trois approches de la programmation par composants parallèles pour le calcul scientifique.

2.4.2 Modèles de programmation par composants parallèles

Cette section présente trois approches des composants parallèles pour le calcul scientifique : elles sont fondées sur des modèles de composants logiciels séquentiels déjà rencontrés à la section 2.2.2.4, à savoir CCM, CCA et Fractal.

2.4.2.1 Composants parallèles GRIDCCM

CCM offre peu de support pour encapsuler des codes parallèles dans ses composants. Des travaux [64] ont montré que CORBA et MPI peuvent cohabiter avec des performances aptes

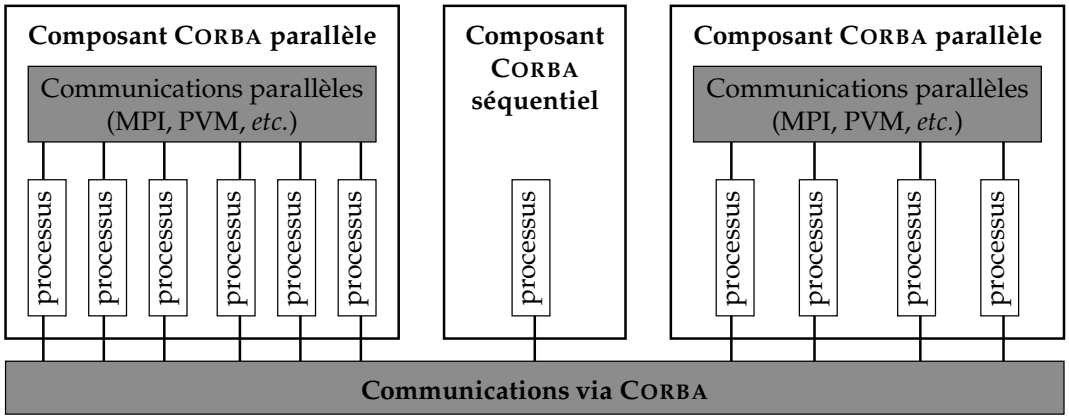


FIG. 2.9 – Application à base de composants séquentiels CCM et parallèles GRIDCCM.

à satisfaire les besoins du calcul scientifique. Cependant, encapsuler un programme parallèle dans un composant CCM classique, et désigner un unique processus comme responsable des communications avec les autres composants, ne passe pas à l'échelle (figure 2.8(a)) : le processus qui joue le rôle d'interface entre le composant parallèle et les autres composants va rapidement devenir un goulet d'étranglement.

Pour régler ce problème, GRIDCCM [142, 139] étend le concept de composants séquentiels CCM à celui de composants parallèles en autorisant les communications directes entre les différents processus des composants. Le parallélisme est vu comme une caractéristique de l'implémentation : la description du composant en IDL3 de CCM n'est pas modifiée par rapport aux composants séquentiels. Ainsi, un composant parallèle peut être remplacé par un composant séquentiel (et vice-versa) sans changer la description des interfaces du composant (en IDL3), ni les outils d'assemblage de composants.

Les communications entre les flots d'exécution au sein d'un composant parallèle donné (figure 2.9) peuvent être réalisées par mémoire partagée ou par passage de messages (MPI, PVM, *etc.*). Les communications entre composants (parallèles ou séquentiels) se font en parallèle via CORBA, avec redistribution éventuelle des données (M×N, *cf.* figure 2.8(b)) si les composants n'ont pas le même degré de parallélisme (égal à 1 pour les composants séquentiels).

2.4.2.2 Composants parallèles CCA

Nous avons déjà présenté XCAT3, une implémentation purement distribuée de CCA, à la section 2.2.2.4. Ccaffeine [14] et SCIRun2 [165] sont deux autres implémentations de CCA, qui supportent les programmes SPMD (MPI, PVM, *etc.*). Les applications Ccaffeine et SCIRun2 sont dites de type SCMD (*Single Component, Multiple Data*, figure 2.10) : chaque composant s'exécute en parallèle sur un certain nombre de processus communs, et chaque processus contient chacun des composants parallèles. En particulier, tous les composants parallèles sont contraints d'avoir le même degré de parallélisme.

Enfin, CCA possède une implémentation des composants parallèles de type similaire à GRIDCCM (figure 2.9), c'est-à-dire autorisant des degrés de parallélisme distincts pour chaque composant : DCA (*Distributed CCA Architecture*, [35]) est fondé sur MPI exclusivement et sur

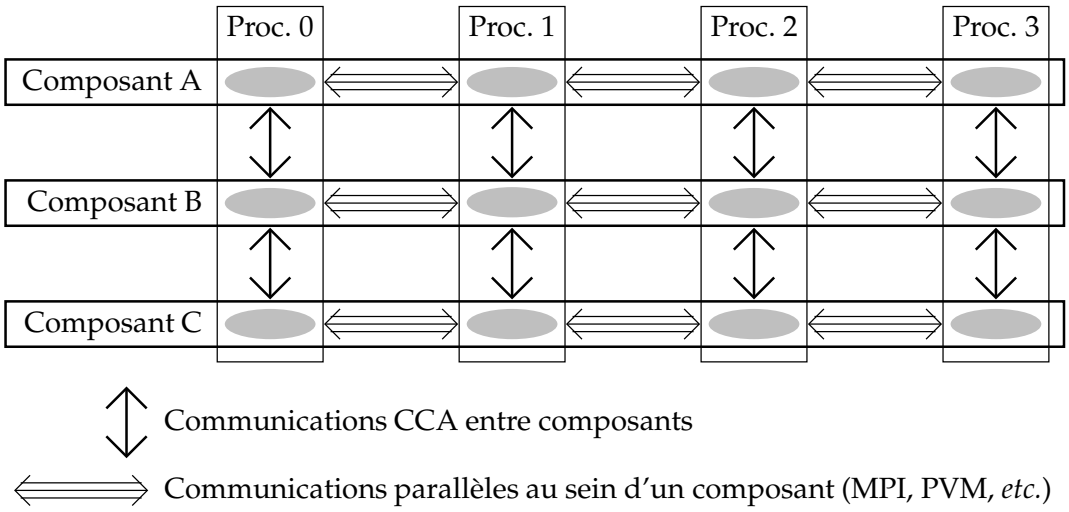


FIG. 2.10 – Application à base de composants parallèles CCA.

PRMI (*Parallel Remote Method Invocation*) pour les appels de méthode à distance. DCA permet la redistribution de données entre des composants parallèles de degrés de parallélisme différents.

2.4.2.3 Composants parallèles ProActive/Fractal

ProActive enrichit les composants Fractal avec le parallélisme [32] en s'appuyant sur ses communications de groupe. Le parallélisme dans ProActive consiste à regrouper plusieurs composants au sein d'un composite, et à diffuser les données destinées au composite vers tous les composants qui le constituent (*multicast*) en une seule opération. Cependant, les communications parallèles entre composants ne sont pas encore gérées dans ProActive : pour chaque composant parallèle, un processus particulier (appelé « *group proxy* ») concentre les communications destinées aux autres composants, et devient donc un goulet d'étranglement.

Pour ce qui est du lancement d'applications, les composants parallèles ProActive héritent des fonctionnalités de déploiement intégrées à ProActive présentées à la section 2.2.2.4.

2.4.3 Discussion

Les applications multi-physiques, qui font coopérer plusieurs codes de calcul numérique intensif, sont des candidats idéaux de la programmation par composants parallèles : la notion de composants apporte la dynamicité (pour la connexion d'un composant de visualisation en cours d'exécution par exemple), le support de l'hétérogénéité, la réutilisabilité des codes, et impose une certaine rigueur dans la structuration de ces applications complexes ; le parallélisme apporte les performances attendues par le calcul scientifique.

Les modèles et implémentations de composants parallèles sont des sujets de recherche très récents : les prototypes n'ont pas encore atteint un degré de maturité élevé. Par exemple, ni ProActive/Fractal, ni la norme CCA ne définissent formellement de modèle de composants

parallèles. Les applications GRIDCCM se déploient par une suite complexe d'opérations manuelles, incluant les techniques de lancement des programmes parallèles (section 2.3) et des programmes CCM (section 2.2). Ainsi, aucun modèle de déploiement d'applications mixtes n'est encore spécifié : comme cette technologie est encore jeune, ces applications sont lancées de manière *ad hoc*, et toute la difficulté repose sur l'utilisateur qui doit lancer ses applications *manuellement*.

2.5 Conclusion

Ce chapitre a présenté trois grandes classes d'applications concurrentes pour le calcul scientifique :

- les applications distribuées à base de composants offrent la dynamicité (connexion d'un programme de visualisation en cours d'exécution par exemple), le support de l'hétérogénéité, la structuration des applications, la réutilisabilité des codes de calcul (qui peuvent s'avérer complexes, et sont souvent développés par des experts du domaine) ;
- les applications parallèles visent la haute performance, en réduisant les temps d'exécution et en acceptant de plus grands volumes de données, pour permettre des calculs toujours plus réalistes et précis ;
- les applications mixtes, qui font coopérer des composants parallèles, bénéficient des avantages propres à la programmation par composants logiciels et à la programmation parallèle, et sont donc idéales pour les applications complexes de couplages de codes parallèles.

Bien que ces trois classes d'applications soient importantes pour le calcul scientifique, les problèmes de déploiement ne sont entièrement résolus pour aucune de ces classes d'applications :

- même si CCM présente le modèle de déploiement le plus abouti, l'utilisateur a toujours la charge de trouver et sélectionner les ressources adéquates pour exécuter son application, d'installer les fichiers (données et exécutables) aux bons emplacements sur les ressources, et de lancer les processus de l'application, éventuellement à distance ;
- MPI et les MVP se sont imposés à une époque où la taille des applications était maîtrisable et où les ressources de calcul étaient relativement simples, donc ces modèles ne définissent pas comment déployer les applications (et les implémentations fournissent des outils sommaires qui fonctionnent différemment les uns des autres) ;
- bien que très prometteuses, les applications mixtes sont encore difficiles à déployer manuellement, et il n'existe pas de modèle de déploiement pour cette classe d'applications.

Parmi les raisons qui peuvent expliquer la timidité à définir des modèles précis de déploiement figure la diversité des infrastructures sur lesquelles les applications peuvent s'exécuter. En effet, cette présentation des classes d'applications s'est faite indépendamment des infrastructures possibles d'exécution. Or le calcul scientifique intensif est très souvent exigeant en puissance de calcul : le chapitre suivant décrit les grilles de calcul comme un environnement de choix pour satisfaire les besoins en ressources des applications.

Chapitre 3

Les grilles de calcul

Sommaire

Le chapitre précédent a présenté différents types d'applications concurrentes adaptées au calcul scientifique sans faire d'hypothèse sur les infrastructures d'exécution qui pourraient les héberger. Le premier objectif de ce chapitre est de définir ce que nous entendons par « grilles de calcul » pour montrer qu'elles sont une infrastructure d'exécution intéressante pour les applications que nous visons. Le second objectif est de mettre en exergue les spécificités des grilles, qu'il faudra prendre en compte pour y déployer automatiquement des applications.

Ce chapitre commence par situer notre vision des grilles de calcul parmi les multiples définitions qui existent, puis il présente les infrastructures matérielles et logicielles attachées aux grilles. Enfin, nous concluons sur l'adéquation des grilles avec les types d'applications présentés au chapitre précédent.

3.1 Définitions et objectifs

Les grilles informatiques sont un domaine en plein essor : elles font l'objet d'un nombre important de travaux de recherche. La multiplicité des travaux sur les grilles informatiques

suivant des angles d'étude différents résulte en une multiplicité des définitions des grilles informatiques.

Cette section commence par présenter quelques unes des visions possibles des grilles, puis elle détaille les éléments distinctifs qui constituent *notre* définition des grilles de calcul.

3.1.1 Origines et évolution

L'émergence des réseaux longue distance (*Wide-Area Network*, WAN) et l'amélioration de leurs performances a permis d'envisager d'exécuter en parallèle des applications sur un grand nombre d'ordinateurs géographiquement répartis sur plusieurs sites. Dans ce contexte, en 1995, les États-Unis lancent un réseau expérimental baptisé *I-Way* [60] (*Information Wide-Area Year*) : il interconnecte dix-sept centres de calcul américains par des liens ATM à 155 Mb/s. Une infrastructure logicielle, appelée *I-Soft* [75] et spécifique à *I-Way*, est réalisée afin de fournir un environnement de programmation parallèle, et de gérer les lancements d'applications, les communications, les transferts de fichiers, et la sécurité.

Ce type de déploiement de réseau à grande échelle entre des super-calculateurs relève du *metacomputing*, un précurseur des grilles informatiques. Le terme de « grille » (*grid*, en anglais) est apparu en septembre 1997 lors d'un séminaire à *Argonne National Laboratory*, aux États-Unis. Il a été répandu plus largement en 1998, à l'occasion de la parution de l'ouvrage de référence [77] publié par Ian Foster[1] et Carl Kesselman[2]. Le terme de « grille » a été choisi par analogie avec le réseau de distribution de l'électricité aux États-Unis (*electric power grid*) : il laisse entendre que la grille informatique pourra fournir de la puissance informatique (calcul, stockage de données, *etc.*) de la même manière que le réseau de distribution électrique fournit la puissance électrique, c'est-à-dire de façon *transparente*.

Dans cet ouvrage [77], Ian Foster et Carl Kesselman tentent une première définition de la « grille de calcul » :

> « *A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.* »

Sentant que cette définition était trop vague, les mêmes auteurs avec Steve Tuecke[3] raffinent [80] cette définition en 2000 : l'introduction de la notion d'« organisation virtuelle » (*virtual organization*, VO) permet de prendre en compte des aspects socio-économiques dans la définition de la grille. Avant de définir ce qu'est une « organisation virtuelle », nous avons besoin d'expliciter ce que nous appelons « ressources informatiques » et « sites de grille ».

Définition 3.1 : ressource informatique — *Une ressource informatique est un élément matériel ou logiciel qui permet de générer, traiter, stocker ou échanger des données informatiques automatiquement.*

Les ressources informatiques peuvent comprendre des liens réseau, des moyens de calcul (ordinateurs, PDA[4], super-calculateurs, *etc.*), des capteurs associés à des convertisseurs numériques (télescopes, *etc.*), des logiciels, des espaces de stockage de données, *etc.* La notion de « ressource informatique » permet de préciser ce que nous appelons un « site de grille ».

[1] *Argonne National Laboratory* (ANL) et *University of Chicago*.
[2] *Information Sciences Institute* (ISI), *University of Southern California*.
[3] *Argonne National Laboratory* (ANL).
[4] *Personal Digital Assistant*.

Définition 3.2 : site d'une grille — *Un site de grille est un ensemble de ressources informatiques qui sont localisées géographiquement dans le même institut, campus universitaire, centre de calcul, entreprise ou chez un individu, et qui forment un domaine d'administration autonome, uniforme et coordonné.*

Ainsi, un site de grille est administré suivant une unique politique d'accès aux ressources : une seule équipe d'administration décide, de manière *autonome*, du matériel et des logiciels rendus disponibles dans la grille par leur site. Les définitions de « ressources informatiques » et « sites de grille » permettent maintenant de préciser ce qu'est une organisation virtuelle.

Définition 3.3 : organisation virtuelle (VO) — *Une organisation virtuelle est un ensemble de sites de grille indépendants, qui se crée et se dissout dynamiquement au gré de leurs besoins en ressources informatiques : les sites d'une organisation virtuelle partagent une partie de leurs ressources informatiques de manière contrôlée.*

Par conséquent, une organisation virtuelle n'est pas figée : elle apparaît et disparaît *dynamiquement* par le biais d'un contrat (souvent à durée déterminée) afin de permettre à différents sites de *mettre en commun* certaines de leurs ressources pour résoudre un problème informatique. Cet objectif ne pourrait pas être atteint par l'un des sites seul, car il nécessite plus de ressources informatiques (puissance de calcul, espace de stockage, *etc.*) qu'un site n'en possède individuellement. Cependant, les ressources sont *partagées de manière contrôlée* : un contrat spécifie quels utilisateurs ont accès à quelles ressources informatiques, à quel moment, pour quelle durée, et dans quelles conditions. Il n'est pas exclu qu'une organisation réelle (institut de recherche, entreprise, *etc.*) fasse partie de plusieurs organisations virtuelles.

Face à l'explosion d'appellations abusives qui employaient le terme de « grille » pour désigner des systèmes qui n'en sont pas, Ian Foster publie une chronique [74] en 2002 pour préciser sa définition d'une grille en trois points. Une grille est un système qui :

1. coordonne des ressources sans contrôle centralisé,
2. en utilisant des protocoles et interfaces standard, ouverts, et génériques (et non spécifiques à une application précise),
3. afin de fournir un service et des fonctionnalités de plus haut niveau que la somme des services offerts par chaque constituant individuel de la grille.

Dans ce contexte d'évolution et de multiplicité des définitions, il est nécessaire de préciser ce que *nous* choisissons d'appeler « une grille informatique ».

Définition 3.4 : grille informatique — *Une grille informatique est l'ensemble des ressources informatiques mises en commun et partagées par les sites d'une organisation virtuelle, ainsi que les moyens de communication entre ces sites, et la spécification de la politique d'accès aux ressources de chaque site par chaque utilisateur.*

Cette définition n'entre pas en contradiction avec les différents recadrages proposés par Ian Foster, Carl Kesselman et Steve Tuecke. Elle est suffisamment générale pour englober plusieurs types de grilles informatiques comme le montre la section suivante.

3.1.2 Multiplicité des grilles informatiques

La section précédente a donné notre définition d'une grille informatique : cette définition est générale, et il existe plusieurs types de grilles comme nous l'illustrons dans cette section. La différence entre ces types de grille porte sur la nature des ressources partagées ou sur leurs objectifs applicatifs. Enfin, nous montrons à quel type de grille nous nous intéressons en particulier, à savoir les grilles de calcul.

Grilles de données (ou grilles de stockage). Les expériences futures de physique des hautes énergies généreront plusieurs téraoctets de données par jour, comme le prévoit le Centre Européen pour la Recherche Nucléaire (CERN, [171]). Des milliers de physiciens, relevant de centaines de centres de recherche, laboratoires et universités du monde entier, auront besoin d'y accéder et de les analyser. Les grilles de données peuvent répondre à ce type de besoin : elles permettent de stocker d'énormes quantités de données et de les « fouiller ».

Grilles de récupération de cycles. À chaque instant, des milliers d'ordinateurs connectés au réseau Internet sont inactifs. L'idée des grilles de récupération de cycles (*scavenging grids*) est de réunir une partie de ces ordinateurs en une organisation virtuelle pour les occuper à exécuter des applications. On parle aussi de *desktop grids* si l'on se restreint à des ordinateurs de bureau, à l'exclusion des serveurs d'exécution et des super-calculateurs. Un des objectifs principaux des grilles de récupération de cycles est le calcul à grande capacité de traitement, qui consiste à faire s'exécuter une application sur le plus grand nombre d'ordinateurs possible.

Grilles de calcul. L'idée des grilles de calcul (*computational grids*) est de rassembler une grande puissance informatique (de calcul et de communication) pour effectuer des calculs intensifs à haute performance. Une grille de calcul met donc plus l'accent sur la puissance totale des moyens de calcul que sur le nombre d'ordinateurs (contrairement aux grilles de récupération de cycles), et elle est souvent équipée de réseaux de communication haute performance. Enfin, les grilles de calcul mettent moins l'accent sur la quantité d'espace de stockage que les grilles de données.

Un réseau haute performance ou un *cluster* (ou « grappe d'ordinateurs ») équipé d'un système de batch ne suffisent pas à constituer une grille de calcul, mais ils peuvent participer à en former une.

Définition 3.5 : système de batch — *Un système de batch est un logiciel qui s'exécute le plus souvent sur le nœud frontal d'un cluster, et par lequel un utilisateur peut soumettre une tâche à exécuter sur les nœuds du cluster.*

Le système de batch fait appel à un ordonnanceur (*scheduler*) pour décider de l'ordre d'exécution des différentes tâches, et pour les placer sur les nœuds du cluster. Les exemples les plus connus de systèmes de batch sont SGE (*Sun Grid Engine*, [228]), PBS (*Portable Batch System*, [217]), LSF (*Load Sharing Facility*, [203]). Un cluster équipé d'un système de batch n'est pas une grille de calcul, car le contrôle des nœuds du cluster est *centralisé*, et tous les nœuds du cluster appartiennent au même domaine d'administration. En revanche, un tel cluster peut entrer dans la composition d'une grille de calcul.

C'est aux grilles de calcul que nous nous intéressons en particulier. Dans la suite de ce document, le terme de « grille » désignera une grille de calcul, telle que nous l'avons définie à la page 37 (définition 3.4), et avec l'objectif du calcul intensif à haute performance mentionné ici.

3.1.3 Caractéristiques des grilles de calcul

Cette section présente quelques caractéristiques importantes des grilles de calcul que nous devrons prendre en considération pour le déploiement d'applications : le partage des ressources, l'hétérogénéité des ressources et des politiques d'administration, la sécurité, la généricité, le facteur d'échelle, et la dynamicité.

3.1.3.1 Partage des ressources

La notion d'organisation virtuelle implique le partage de ressources informatiques au sein de la grille. Une grille se distingue d'Internet, car elle ne se contente pas de faire partager de l'information, mais elle fait aussi partager de la puissance de calcul, de l'espace de stockage, des bases de données, des logiciels, *etc.* Internet n'est qu'un élément constitutif de certaines grilles (un réseau), utile pour la communication entre sites.

3.1.3.2 Hétérogénéité

L'autonomie d'administration des différents sites d'une grille implique souvent une forte hétérogénéité, tant pour les ressources informatiques de la grille que pour les politiques d'accès à ces ressources. C'est cette hétérogénéité qui rend nécessaire l'utilisation de protocoles standard et ouverts.

Hétérogénéité des ressources informatiques.

Moyens de calcul. En termes de puissance de calcul, on peut aussi bien trouver des super-calculateurs que des ordinateurs de bureau (PC), des serveurs d'exécution, des stations de travail, *etc.* Ce point distingue le calcul sur grille du *metacomputing*, qui ne fait intervenir que des super-calculateurs.

Architectures. En termes d'architecture matérielle, les ordinateurs peuvent être équipés de différents types de processeurs : PowerPC, compatibles i386, des Alphas, des Mips, *etc.*

Logiciels. En termes d'installation logicielle, les ordinateurs peuvent avoir différents systèmes d'exploitation avec une version précise (AIX 5.3, IRIX 6.5, Solaris 9, Linux 2.6.10, Windows XP, *etc.*). Les logiciels disponibles et leurs versions peuvent également être différents et installés à des endroits variés (compilateurs, bibliothèques de calcul, *etc.*).

Réseaux. En termes de réseaux d'interconnexion entre les ordinateurs, les liens de communication peuvent avoir des débits, latences, gigues, taux de pertes différents.

Hétérogénéité des politiques d'accès aux ressources. Chaque site d'une grille décide de façon autonome et indépendante des politiques :
- d'authentification, en sélectionnant une ou plusieurs méthodes de connexion (telnet, SSH, *etc.*), et éventuellement des algorithmes de chiffrement des communications ;
- d'autorisation d'accès aux ressources, afin de déterminer quels utilisateurs possèdent quels droits (lecture, écriture, effacement) sur chaque ensemble de données ;
- d'attribution des noms d'utilisateur.

3.1.3.3 Sécurité

Une organisation virtuelle est constituée d'organisations réelles qui se font une confiance limitée. De plus, la mise en place d'une grille doit respecter (et non infléchir) les politiques d'accès aux ressources de chaque site. Ainsi, la sécurité et ses mécanismes de mise en œuvre sont des problématiques cruciales des grilles.

La sécurité concerne l'authentification mutuelle des utilisateurs et des ressources : ces mécanismes consistent à prouver qu'un utilisateur est bien qui il prétend être, et qu'une ressource est effectivement ce qu'elle prétend être (service d'information, serveur d'exécution, *etc.*). La sécurité concerne également les questions d'autorisation définies au paragraphe 3.1.3.2, et le chiffrement des données. Le chiffrement sert non seulement pour les informations en transit sur les liens de communication, mais aussi pour celles qui sont stockées dans les mémoires et les disques. Il évite la lecture des données secrètes, et protège leur intégrité en empêchant leur modification sans autorisation. Enfin, la sécurité peut impliquer la nécessité de comptabiliser les accès (en nombre et en durée) des utilisateurs aux ressources (*accounting*, en anglais).

Cet accent mis sur la sécurité *coordonnée* entre différents sites *autonomes* qui se font une confiance limitée distingue les grilles de la plupart des autres systèmes distribués à grande échelle.

3.1.3.4 Généricité

Dans les grilles, la généricité s'entend pour le type d'applications qui peut s'y exécuter, ainsi que pour les protocoles qui permettent d'utiliser les ressources de la grille.

Certes, une grille donnée peut être mise en place pour exécuter une classe d'applications bien précise. Mais elle existe pour une durée limitée dans le temps, et il n'est peut-être pas souhaitable de développer des protocoles spécifiques à chaque grille et pour chaque classe d'applications. Ainsi, les protocoles d'accès aux ressources des grilles doivent être génériques, pour pouvoir être réutilisés dans d'autres grilles qui visent d'autres applications, mais aussi pour assurer l'intéropérabilité entre les sites d'une grille.

3.1.3.5 Grande échelle

La nécessité de recourir à des organisations virtuelles a plusieurs origines :
- une organisation réelle ne possède pas forcément le savoir-faire et les codes de calcul dont elle a besoin ;
- une organisation réelle peut manquer de ressources financières pour satisfaire à elle seule ses besoins informatiques (ponctuels ou non).

Ainsi, plusieurs organisations réelles peuvent mettre en commun leurs ressources pour satisfaire leurs besoins informatiques. Ce partage est d'autant plus nécessaire que les besoins sont ponctuels, et qu'il n'est donc pas rentable d'investir dans ces ressources informatiques (matérielles ou logicielles).

La mise en commun de ressources de plusieurs sites distribués géographiquement peut alors donner naissance à des systèmes à grande échelle, tant du point de vue de la quantité de ressources que de leur dispersion spatiale.

3.1.3.6 Dynamicité et tolérance aux défaillances

Il est vrai que les défaillances matérielles et logicielles font partie intégrante des systèmes distribués à grande échelle : lien réseau coupé, ordinateur qui tombe en panne, programme non conforme aux spécifications, *etc.* Sans nier leur importance, nous ne nous focalisons pas sur la dynamicité (et l'adaptabilité qu'elle nécessite), ni sur la tolérance aux défaillances dans les grilles de calcul. En effet, traditionnellement, la dynamicité des grilles de calcul n'est pas (encore) un sujet de préoccupation aussi dominant que pour les systèmes pair-à-pair. Les déconnexions sans préavis sont la règle dans le pair-à-pair, tandis que les grilles de calcul sont beaucoup plus stables : chaque site d'une grille est maintenu par une équipe professionnelle d'administration.

3.1.4 Discussion

3.1.4.1 Objectifs à long terme des grilles de calcul

Outre la réduction des coûts d'investissement en ressources informatiques et une meilleure régulation de leur utilisation, un des objectifs *capitaux* des grilles de calcul est de *fournir de la puissance informatique de manière transparente*. Cet objectif est hérité du réseau de distribution de l'électricité, dont la grille tire son nom.

Dans le cas des grilles de calcul, la *transparence* implique que l'utilisateur de la puissance informatique doit pouvoir se contenter de lancer son application sur la grille pour obtenir un résultat, sans avoir à se soucier de savoir

- sur quels ordinateurs l'application s'exécute,
- où sont localisés ces ordinateurs,
- de quel type ils sont (super-calculateurs, clusters, *etc.*),
- quels sont leurs systèmes d'exploitation et leurs architectures,
- *etc.*

Par analogie avec le courant électrique, quand on branche un appareil électrique dans une prise, on ne se pose pas la question de savoir où est la centrale qui produit l'énergie qu'on consomme, ni de quelle manière cette électricité est produite (hydraulique, solaire, éolienne, marémotrice, nucléaire, thermique, *etc.*).

Pour poursuivre l'analogie, on doit pouvoir exécuter n'importe quel type d'application sur une grille de calcul, de même qu'on peut brancher (presque) n'importe quel appareil électrique dans une prise de courant (aspirateur, grille-pain, *etc.*). À terme, l'idéal serait de pouvoir utiliser une grille de calcul comme un ordinateur unique et générique, en toute sécurité, simplement, et efficacement.

3.1.4.2 Nouveau défi scientifique et petite révolution

Parmi les multiples définitions possibles d'une grille, nous choisissons de retenir les principales caractéristiques des grilles de calcul : au sein d'une organisation virtuelle, des ressources informatiques hétérogènes sont partagées à grande échelle, de manière coordonnée et sécurisée, afin d'y exécuter des applications génériques.

Les grilles sont des systèmes différents des autres systèmes distribués à grande échelle : la puissance de calcul, le stockage, les logiciels, *etc.* ne sont plus des objets qu'on *possède*, mais auxquels on s'*abonne* ! En cela, les grilles sont une petite révolution sociale dans le monde de l'informatique. Une fois que l'objectif de transparence d'utilisation des grilles sera atteint, elles seront un système informatique comme n'importe quel autre ordinateur : cet objectif de transparence est un nouveau défi scientifique.

3.2 Infrastructures matérielles des grilles de calcul

La section précédente a défini les grilles de calcul de façon générale et relativement abstraite. Cette section présente quelques infrastructures matérielles typiques des grilles de calcul, et montre que ces systèmes sont, de nos jours, une réalité concrète à travers des exemples de grilles existantes. L'objectif est de souligner quelques problématiques qui sont liées aux infrastructures matérielles des grilles, et que nous aurons à résoudre pour automatiser le déploiement d'applications sur ce type de système.

3.2.1 Architectures matérielles des grilles de calcul

De quoi une grille de calcul est-elle constituée ? Une grille de calcul se compose d'un ensemble de nœuds reliés par des liens réseau comme l'illustrent les deux paragraphes suivants.

3.2.1.1 Nœuds de calcul et de stockage

Les nœuds d'une grille de calcul sont ses moyens de stockage de données informatiques, ses moyens de calcul, des dispositifs de visualisation (écrans de réalité virtuelle immersive). Les nœuds peuvent aussi désigner des instruments de mesure ou des capteurs, tels que des télescopes, des senseurs météorologiques, mais on rencontre plus rarement ce type de ressource dans les grilles de calcul génériques qui sont déployées de nos jours (*cf.* section 3.2.2). Parmi les moyens de calcul, on trouve le plus souvent des ordinateurs de bureau (des PC), des stations de travail, des serveurs d'exécution, des super-calculateurs (calculateurs parallèles à mémoire partagée tels que des SGI Origin 3000 [230] ou des SMP, calculateurs vectoriels tels que des Cray XT3 [173]), des clusters, ainsi que, dans une moindre mesure, des PDA et des ordinateurs portables.

3.2.1.2 Réseaux de communication

Les nœuds d'une grille de calcul, quelle que soit leur nature, doivent être interconnectés par un ou plusieurs réseaux de communication pour coopérer en échangeant des données. Les liens réseau font partie à part entière des ressources informatiques d'une grille de calcul.

Beaucoup de grilles ont tout ou partie de leurs nœuds reliés à Internet (où la latence peut dépasser 300 ms), mais ce n'est pas nécessaire. Les différents sites d'une grille peuvent être interconnectés par un réseau privé, tel que l'épine dorsale de TeraGrid [233] aux États-Unis (débit : 40 Gb/s), ou le réseau VPN[5] fourni par RENATER[6] à Grid'5000 [185] en France (débit : 10 Gb/s).

Les nœuds au sein d'un site peuvent être reliés par des réseaux locaux (*Local-Area Network*, LAN), de type FastEthernet (débit : 100 Mb/s, latence de l'ordre de 90 μs) ou Gigabit Ethernet (débit : 1 Gb/s), ou bien encore par des réseaux sans fil (débit : 54 Mb/s par exemple). Les nœuds d'un cluster peuvent être interconnectés par un réseau haute performance (*System-Area Network*, SAN) tel que Myrinet [207, 38] (débit : 2 Gb/s, latence de l'ordre de la microseconde), Quadrics [220] (débit : plus de 6 Gb/s, latence de l'ordre de la microseconde), ou InfiniBand [196].

Ainsi, une grille peut comporter une grande variété de technologies d'interconnexion réseau, et toute une hiérarchie de réseaux en termes d'étendue géographique, et en termes de performances des communications (débit, latence, *etc.*). Des réseaux longue distance (*Wide-Area Network*, WAN) relient les sites de la grille ; les nœuds à l'intérieur de chaque site peuvent être interconnectés par des réseaux locaux (LAN) ou par des réseaux haute performance (SAN) au sein d'un cluster. À tous les niveaux, les liens réseau peuvent être redondants, ce qui permet à deux nœuds de communiquer par une technologie réseau ou par une autre.

3.2.2 Exemples de grilles de calcul

La section précédente a donné une description générale des types de ressources qui constituent une grille de calcul. Pour illustrer cette description générale, cette section présente rapidement quelques exemples de grilles informatiques, afin de montrer que ce sont des systèmes concrets, en plein essor, et à l'origine d'une véritable activité économique : IBM [194] et Sun Microsystems [232] se sont intéressés très tôt aux grilles de calcul.

TeraGrid. TeraGrid [233] est un projet qui vise à mettre en place la plus grande et la plus puissante grille de calcul scientifique du monde. TeraGrid comporte des dispositifs et environnements de visualisation à haute résolution (réalité virtuelle immersive, *etc.*), des serveurs de calcul parallèle, des serveurs de stockage de données parallèles. Les neuf sites de TeraGrid sont répartis à travers les États-Unis, et sont reliés par un réseau optique longue distance de débit 40 Gb/s. Les neuf sites cumulent plus de 30 Tflops[7] de puissance de calcul, plus de 3 PB[8] de capacité de stockage de données, plus de 7000 processeurs. Les moyens de calcul et de communication sont très variés :

- architectures matérielles : Itanium IA64, SGI Altix [229], Power4 IBM, HP/Compaq, *etc.*
- systèmes d'exploitation : Linux, Tru64, *etc.*

[5] *Virtual Private Network*.

[6] Réseau National de Télécommunications pour la Technologie, l'Enseignement et la Recherche, [221].

[7] flops = *floating point operations per second*, unité de mesure de la puissance de calcul maximale d'un système informatique. 1 Tflops = 10^{12} flops.

[8] PB = *PetaByte* = 10^{15} octets.

- types de calculateurs : PC bi-processeurs ou quadri-processeurs, serveurs CC-NUMA[9] à mémoire partagée, machines symétriques SMP à mémoire partagée, *etc.*
- réseaux des clusters (SAN) : Myrinet [207, 38], Quadrics [220], Gigabit Ethernet, *etc.*

Chacun des neuf sites de TeraGrid administre ses ressources et ses utilisateurs de manière assez indépendante (par exemple, les noms d'utilisateur Unix sont différents d'un site à un autre pour un même individu).

Projets européens. L'Europe est également active en matière de grilles de calcul, poussée notamment par le CERN (Centre Européen pour la Recherche Nucléaire, [171]) qui mettra en service en 2007 le LHC (*Large Hadron Collider*). Ainsi, les expériences de physique des hautes énergies au CERN produiront environ 10 pétaoctets de données par an à partir de 2007. Parmi les projets européens, on peut citer **DataTAG** (*Data TransAtlantic Grid*, [174]), **EGEE** (*Enabling Grids for E-sciencE*, [178]), **DEISA** (*Distributed European Infrastructure for Supercomputing Applications* [**?**]), *etc.*

Grid'5000. Grid'5000 [185] est une initiative dont l'objectif est de déployer en France 5 000 processeurs interconnectés pour mener des expériences de recherche sur les grilles. Les ressources de Grid'5000 sont réparties sur neuf sites à travers la France métropolitaine. Ces sites sont reliés par un réseau privé virtuel (VPN) à 1 Gb/s. Chaque site héberge un ou plusieurs clusters aux architectures matérielles et aux systèmes d'exploitation variés : Linux sur Intel Xeon 32 bits ou sur AMD Opteron 64 bits, Darwin sur Apple MacIntosh PowerPC, *etc.* Les réseaux SAN des clusters sont également diversifiés : Gigabit Ethernet, Myrinet, InfiniBand, *etc.*

3.2.3 Discussion

Les grilles de calcul sont donc bien une réalité tangible : les volumes de données produites par les expériences de physique, ainsi que les calculs scientifiques de simulation et modélisation en physique, biologie et sciences de la terre conduisent naturellement à un besoin massif en grilles de calcul. En pratique, les grilles de calcul sont souvent constituées de fédérations de clusters qui sont dispersés géographiquement (figure 3.1). En effet, les clusters supplantent de plus en plus les super-calculateurs, car ils sont plus économiques, plus extensibles et d'usage plus générique.

Comme la puissance de calcul et de stockage des grilles s'obtient par le partage d'un très grand nombre de ressources qui appartiennent à des sites autonomes, les grilles sont particulièrement hétérogènes et la sécurité doit être assurée entre ces sites. Par conséquent, les grilles sont des environnements complexes à utiliser. Ainsi, il est nécessaire de bien connaître les détails de l'infrastructure des grilles (systèmes d'exploitation, architectures matérielles, *etc.*) pour pouvoir y lancer des applications. Pour aider à vaincre la complexité d'utilisation des ressources de grille, des logiciels sont nécessaires, comme l'illustre la section suivante.

[9] CC-NUMA : *Cache Coherent, Non-Uniform Memory Access*, machine multi-processeurs, où chaque processeur possède son propre banc mémoire auquel il peut accéder directement et très rapidement ; chaque processeur peut également accéder aux bancs mémoire des autres processeurs de la machine, mais de manière un peu moins rapide.

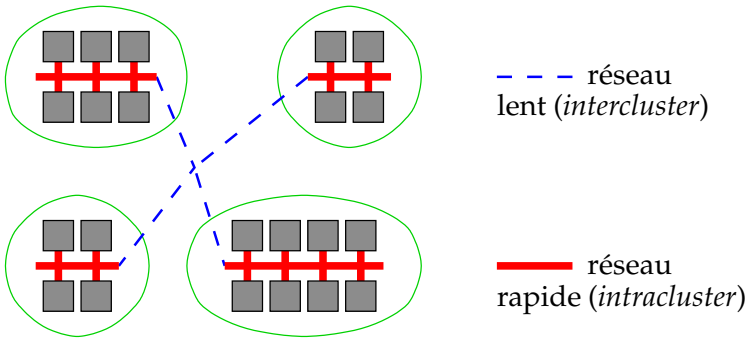


FIG. 3.1 – Hétérogénéité des réseaux dans une fédération de clusters.

3.3 Infrastructures logicielles des grilles de calcul

Comme nous venons de le voir dans la section précédente, les ressources d'une grille sont hétérogènes (du point de vue du matériel, comme des politiques d'administration), très nombreuses, distribuées géographiquement, et elles nécessitent des garanties en matière de sécurité. Les grilles sont donc des environnements particulièrement complexes.

Une des clés du succès des grilles de calcul est la suite logicielle qui facilite l'accès à leurs diverses ressources de manière sécurisée. Cette section définit les rôles d'une telle suite logicielle, puis en présente quelques exemples. Enfin, nous soulignons la nécessité des suites logicielles, tout en remarquant qu'elles ne résolvent pas tous les problèmes, et qu'elles ne suffisent pas actuellement pour utiliser les grilles de façon aussi transparente que les objectifs du paragraphe 3.1.4.1 le laissent espérer.

3.3.1 Définition, rôle et nécessité

Pour approcher l'objectif de transparence d'utilisation des grilles de calcul, on ne peut pas laisser un utilisateur qui voudrait lancer son application seul face aux ressources matérielles brutes de la grille. Entre

– l'application qui fait un calcul utile (ou « programme métier »),

– et les multiples systèmes d'exploitation et politiques d'accès des ressources des grilles,

se trouve un « intergiciel d'accès aux ressources de la grille ».

Définition 3.6 : intergiciel (*middleware*) d'accès à la grille — *Un intergiciel* (middleware) *d'accès aux ressources de grille (ou plus simplement « intergiciel d'accès à la grille ») est une suite logicielle qui se trouve entre l'application de l'utilisateur de la grille (programme métier) et les ressources informatiques matérielles et logicielles de la grille.*

Dans cette définition, les « ressources informatiques » correspondent à la définition 3.1 (page 36) : elles comprennent les systèmes d'exploitation des moyens de calcul, les logiciels, compilateurs et bibliothèques qui y sont installés, ainsi que leurs systèmes de batch (*cf.* définition 3.5 page 38) pour la soumission des tâches.

Le rôle de cet intergiciel est de faciliter l'utilisation de la grille, en donnant une vision plus uniforme et intégrée de ses ressources hétérogènes, et en assurant la sécurité des données

et des communications. Les rôles principaux des intergiciels d'accès à la grille peuvent se résumer de la façon suivante.

- Gestion des données : stockage de grandes quantités de données distribuées, localisation des données, transfert fiable et efficace des données, *etc.*
- Gestion des ressources : soumission de tâche à exécuter, contrôle d'une tâche en cours d'exécution (la suspendre, la redémarrer, l'annuler, interroger son état[10], *etc.*).
- Service d'information : obtenir des informations (réparties, statiques ou dynamiques) sur les ressources informatiques de la grille, telles que leur disponibilité, la charge processeur des moyens de calcul, la topologie et la performance des réseaux de communication, la nature des systèmes d'exploitation et leurs versions, les tailles des espaces de stockage disponibles, *etc.*
- De manière transversale aux trois rôles précédents, l'intergiciel doit assurer la sécurité des données (confidentialité, intégrité), la sécurité d'accès aux ressources (authentification et autorisation), et la sécurité des communications (chiffrement).

Les intergiciels d'accès à la grille sont nécessaires pour fournir une vision globale et unifiée d'une grille, ainsi que pour mettre en œuvre le partage des ressources entre les sites de façon sécurisée. En effet, la mise en place d'une grille ne doit pas dépendre de la volonté des administrateurs de chaque site d'uniformiser les systèmes d'exploitation, les compilateurs, bibliothèques et logiciels installés, ou les politiques d'accès aux ressources. Pour qu'une grille puisse se mettre en place de façon souple, l'autonomie des sites doit être respectée, et il appartient à l'intergiciel d'accès à la grille de combler les écarts entre les disparités de chaque site qui la compose.

Discussion. Les rôles des intergiciels d'accès aux ressources des grilles semblent proches de ceux des systèmes d'exploitation pour les grilles, tels GridOS. GridOS [187, 132] fournit des services de système d'exploitation pour supporter le calcul sur grilles. L'objectif de GridOS est de simplifier la mise en œuvre des intergiciels d'accès aux grilles de calcul grâce à des modules noyau permettant les entrées-sorties haute performance, la gestion des ressources et des processus, *etc.*

Ainsi, GridOS ne fournit pas (encore ?) *tous* les services des intergiciels pour les grilles, mais offre seulement quelques services de base pour simplifier l'implémentation des intergiciels et améliorer leurs performances.

3.3.2 Exemples d'intergiciels d'accès aux grilles de calcul

Cette section présente quelques exemples d'intergiciels d'accès aux grilles de calcul, ainsi que leurs principales caractéristiques. Elle insiste plus sur la boîte à outils Globus, car une partie de nos travaux se sont appuyés sur cet intergiciel.

3.3.2.1 Globus Toolkit™

Globus [184, 76, 78] est un intergiciel sous forme de boîte à outils logicielle (implémentée sous la forme de bibliothèques et de petits programmes en ligne de commande) qui permet

[10] Exemples d'états possibles : en cours d'exécution, en attente de ressources, terminée avec succès, suspendue, avortée, *etc.*

d'accéder aux ressources d'une grille. L'essentiel de cet intergiciel est développé à *Argonne National Laboratory* (États-Unis) par une équipe de recherche dirigée par Ian Foster. Le Globus Toolkit est l'intergiciel d'accès aux grilles le plus utilisé à travers le monde, comme l'illustrent des projets tels que GriPhyN [191], DOE Science Grid (initiative du département de l'énergie américain, [177]), DataTAG (projet européen, [174]), TeraGrid [233], *etc.*

L'infrastructure logicielle de **sécurité** de Globus est GSI (*Grid Security Infrastructure*, [81]) : elle permet l'authentification sécurisée et le chiffrement des communications. GSI est fondé sur le chiffrement à clef publique (certificats X.509) et le protocole de communication SSL (*Secure Sockets Layer*). Un utilisateur de Globus doit initialiser un *proxy* (un certificat signé par l'utilisateur) en tapant un mot de passe long. Ce certificat permet d'authentifier l'utilisateur auprès des ressources lors de la soumission de tâches à exécuter sur la grille. Ce *proxy* permet plusieurs soumissions tandis que l'utilisateur n'a eu besoin de taper son mot de passe qu'une seule fois (*single sign-on*). GSI est complété par CAS (*Community Authorization Service*, [134]), qui permet à chaque site de la grille de contrôler l'accès à ses ressources à gros grain (groupes de ressources, groupes d'utilisateurs) et à grain fin (ressources individuelles, utilisateurs individuels).

GRAM (Globus Resource Allocation Manager, [56]) est le module de **gestion des ressources** de la grille : il supporte la soumission et le contrôle de tâches sur des ressources distantes, et il utilise GSI pour l'authentification mutuelle des ressources et des utilisateurs. Les utilisateurs sont identifiés par un certificat X.509 qui est converti sur chaque ressource en un nom d'utilisateur local (`username` Unix par exemple). Pour soumettre à Globus une tâche à exécuter, l'utilisateur doit écrire un script RSL (*Resource Specification Language*, [224]) qui indique, à bas niveau, le ou les fichiers exécutables à lancer, et sur quelles ressources de la grille ces exécutables doivent être lancés. RSL permet aussi de spécifier l'environnement des exécutables (répertoire de travail, variables d'environnement), et les fichiers à rapatrier sur chaque ressource de calcul. GRAM offre à l'utilisateur une interface unique pour exécuter des tâches sur des ressources distantes qui peuvent utiliser différents systèmes de batch, systèmes d'exploitation, *etc.* Le daemon Globus du service GRAM s'appelle le « *GateKeeper* » : c'est à ce daemon que les scripts RSL sont soumis, et c'est ce *GateKeeper* qui s'interface avec les systèmes de batch locaux.

Le module de **gestion des données** dans Globus est représenté par :
- GASS (*Global Access to Secondary Storage*, [36]), qui permet de transférer les fichiers d'entrée vers les nœuds de calcul avant exécution, et de rapatrier les fichiers de sortie après exécution ;
- GridFTP [155, 156], un protocole de transfert de fichiers haute performance, sécurisé (grâce à GSI), fiable, optimisé pour des réseaux de communication longue distance à fort débit ;
- RFT (*Reliable File Transfer*, [222]) permet de contrôler le transfert de fichiers entre deux serveurs GridFTP distants : avec RFT, il est possible d'initier des transferts de fichiers, puis de quitter sa session, et de reprendre sa session depuis une autre machine en reprenant le contrôle du transfert qui s'est poursuivi pendant toute la durée où la session de l'utilisateur était interrompue ;
- RLS (*Replica Location Service*, [143, 223]) permet de convertir des noms logiques d'éléments de données en des pointeurs (URL) vers les localisations physiques de ces données, éventuellement répliquées.

Enfin, le **service d'information** sur la grille et ses ressources est le MDS (*Monitoring & Discovery System*, [55, 204]). Il permet le stockage d'informations distribuées (statiques et dy-

namiques) sur les ressources de la grille, organisé de façon hiérarchique et extensible. Le MDS inclut les mécanismes d'authentification des ressources et des utilisateurs de GSI.

3.3.2.2 UNICORE

UNICORE (*UNiform Interface to COmputing REsources*, [234]) est un projet allemand dont l'objectif est de permettre aux scientifiques d'exploiter différents centres de calcul disséminés à travers l'Allemagne comme un seul calculateur. UNICORE, et son successeur UNICORE Plus, proposent un logiciel intégré avec des interfaces graphiques, contrairement à Globus qui implémente une boîte à outils générique. Bien que l'approche soit différente, UNICORE propose les mêmes fonctionnalités que Globus :

- la sécurité : authentification des utilisateurs et des ressources, mécanismes de contrôle d'accès (autorisation), basés sur les certificats X.509 et la couche de communication SSL (comme Globus) ;
- la gestion des ressources : méthode uniforme de soumission de tâches à exécuter sur des moyens de calcul distants et hétérogènes, contrôle des tâches en cours d'exécution ;
- gestion des données : transfert efficace de données entre les sites de la grille ;
- service d'information sur les caractéristiques des ressources de la grille.

Le projet GRIP (*GRid Interoperability Project*, [190]) est un projet européen qui vise à développer l'intéropérabilité entre Globus et UNICORE, en influençant la mise au point de normes internationales sur les grilles, notamment auprès du GGF (*Global Grid Forum*, [183]).

3.3.2.3 Legion

Le projet Legion [202, 90] a vu le jour à l'université de Virginie (États-Unis) en 1993. Son objectif est de donner la vision d'un unique super-ordinateur virtuel en agrégeant les ressources de la grille. L'originalité de Legion est de proposer une approche complètement orientée objet : toutes les ressources, matérielles et logicielles, fichiers et applications, sont des objets qui appartiennent à des classes. Legion vise à offrir la sécurité, la haute performance via le parallélisme (support de langages parallèles, ainsi que de bibliothèques parallèles telles que MPI, PVM), l'ordonnancement des tâches, la gestion des ressources distribuées, le respect de l'autonomie des sites de la grille, le passage à l'échelle avec la quantité de ressources, la tolérance aux défaillances des nœuds, *etc.*

3.3.3 Discussion

Les exemples d'intergiciels d'accès à la grille que nous venons de décrire jouent bien leur rôle en facilitant l'utilisation sécurisée des grilles. Ils offrent une vision unifiée des ressources géographiquement distribuées et aux méthodes d'accès variées, ils fournissent des outils de base pour transférer des données efficacement et en toute sécurité, *etc.* Ces fonctionnalités sont aussi nécessaires que le sont les langages de programmation de haut niveau par rapport aux langages d'assemblage. Nous remarquons également que les intergiciels n'imposent pas de modèle de programmation particulier : les rôles des intergiciels de grilles et des modèles de programmation sont clairement séparés.

Cependant, même si les intergiciels offrent les fonctionnalités de base d'accès aux grilles, ils ne sont pas suffisants pour rendre l'utilisation des grilles aussi transparente que les objectifs

du paragraphe 3.1.4.1 le laissent espérer. En effet, s'il n'est plus nécessaire d'être un expert en grilles de calcul et de connaître les détails techniques relatifs à chacune des ressources, il est en revanche indispensable d'être un expert en intergiciels d'accès aux grilles et d'avoir une bonne connaissance de l'architecture de la grille sur laquelle on veut lancer une application. À l'heure actuelle, il n'est toujours pas possible de « brancher son application » depuis son ordinateur de bureau pour qu'elle s'exécute *quelque part* comme on brancherait tout appareil électrique sans se soucier des questions techniques de la production et du transport de l'électricité.

Enfin, on peut regretter l'absence de norme d'intéropérabilité entre les intergiciels qui soit unanimement acceptée : les grilles sont encore estampillées « Globus », « UNICORE », ou bien « Legion », *etc.* Heureusement, depuis 2002, il existe une activité qui vise à définir des normes pour la grille, appelée OGSA [79]. Le GGF est en charge de faire aboutir cette norme, influencé par le projet GRIP qui vise à développer l'intéropérabilité entre Globus et UNICORE.

3.4 Conclusion

Ce chapitre a défini ce que nous entendons par « grilles de calcul », et a mis en exergue les spécificités des grilles qu'il faut prendre en compte pour y déployer des applications.

Parmi les multiples définitions possibles d'une grille de calcul, nous avons choisi de retenir celle qui les présente comme l'ensemble des ressources informatiques partagées par les sites d'une organisation virtuelle. Les principales caractéristiques des grilles de calcul sont :

- l'hétérogénéité des ressources matérielles (architectures et puissance des ordinateurs, topologies et performances des réseaux de communication, *etc.*) et logicielles (systèmes d'exploitation, bibliothèques installées, *etc.*) ;
- leur grande échelle, tant en terme de distribution géographique qu'en terme de quantité de ressources partagées ;
- la généricité des types d'applications qu'on doit pouvoir y exécuter : même s'il est naturel que des grilles se forment pour étudier un domaine scientifique particulier, il n'est pas souhaitable de concevoir des outils spécifiques à chaque domaine d'étude, mais il est préférable de prévoir des outils génériques et réutilisables à moindres frais. Enfin, nous remarquons que les intergiciels tels que Globus Toolkit fournissent des fonctionnalités pour accéder aux ressources d'une grille, mais ils n'imposent pas de modèle de programmation particulier : les rôles des intergiciels de grilles et des modèles de programmation sont clairement séparés.

L'un des objectifs originaux des grilles de calcul et de stockage est de *fournir de la puissance informatique de manière transparente*. Comme le montrent les nombreux exemples de grilles qui existent réellement à travers le monde, la puissance de calcul et de stockage est effectivement disponible. Il est donc raisonnable de vouloir y exécuter les applications de calcul scientifique qui nous intéressent (distribuées, parallèles, ou mixtes), grâce notamment à l'amélioration des performances des réseaux longue distance (WAN).

Ce chapitre et le précédent ont présenté les applications et les infrastructures d'exécution qui nous intéressent. Le chapitre suivant va poser le problème de savoir *comment déployer* ces applications sur des grilles de calcul : il s'attachera à montrer la *complexité du déploiement* manuel si l'on s'en tient à l'état de l'art dans ce domaine. L'objectif sera de souligner la nécessité d'*automatiser* le processus de déploiement d'applications sur les grilles de calcul.

Chapitre 4

Déploiement d'applications sur des grilles de calcul

Sommaire

Le chapitre 2 a illustré la diversité des types d'applications concurrentes pour le calcul scientifique, et a montré que les modèles de programmation utilisés offrent un support variable et limité pour le lancement des applications. Ce défaut de support du déploiement s'explique en partie par la volonté de laisser les modèles de programmation indépendants des infrastructures d'exécution. D'autre part, le chapitre 3 a montré que les grilles de calcul sont une infrastructure de choix pour les applications concurrentes de calcul scientifique, grâce notamment à la puissance de calcul qu'elles peuvent fournir. Cependant, les intergiciels qui permettent d'accéder aux ressources des grilles sont variés et complexes d'utilisation. L'objectif de ce chapitre est de montrer la difficulté actuelle à déployer les applications concurrentes de calcul scientifique, pour en déduire la nécessité d'automatiser le processus de déploiement des applications sur les grilles de calcul.

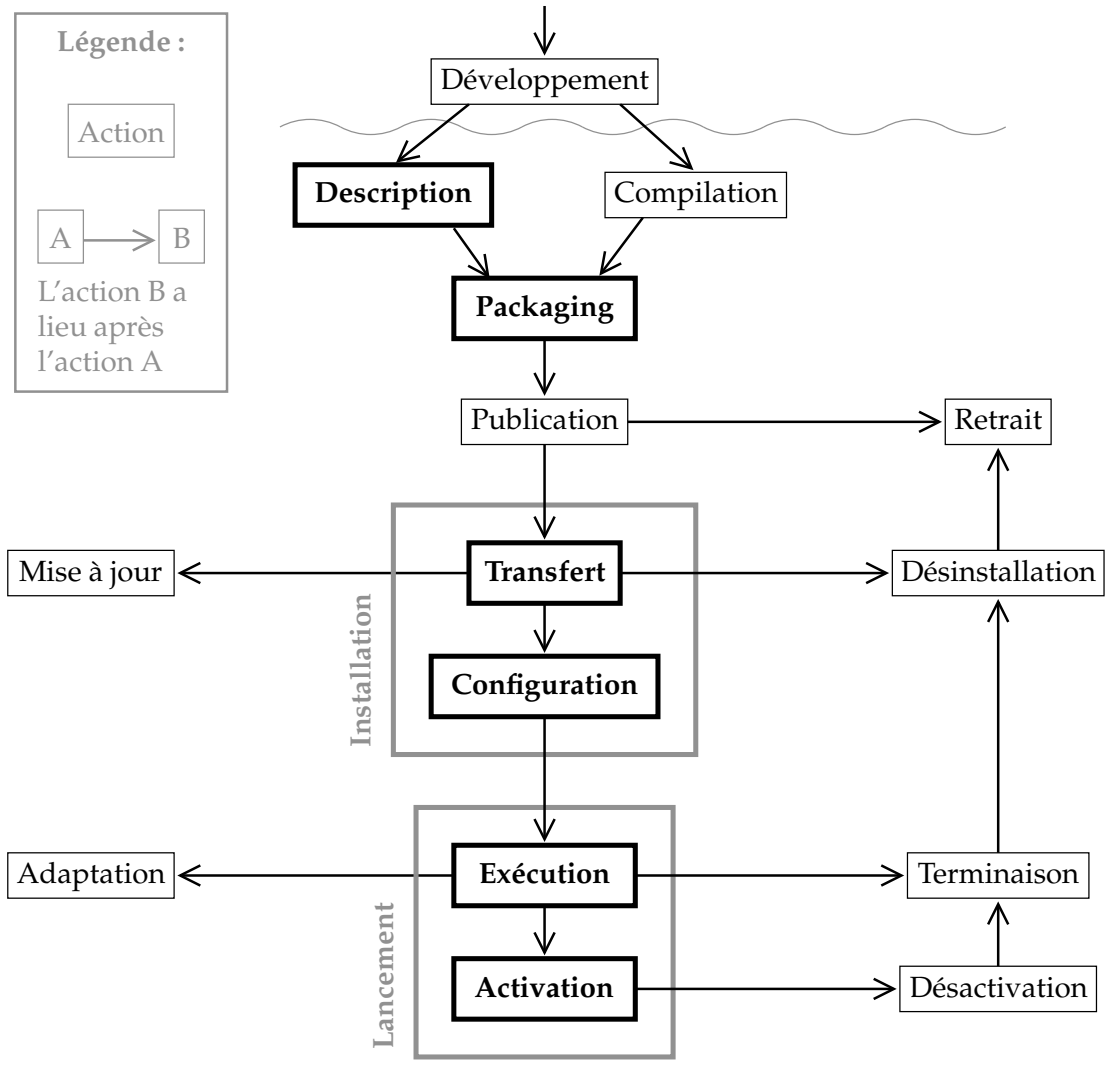


FIG. 4.1 – Une partie du cycle de vie d'une application après développement. Les étapes sur lesquelles nous avons travaillé sont représentées en gras.

Ce chapitre commence par définir ce que nous entendons par « déploiement d'applications », pour bien délimiter le problème que nous avons cherché à résoudre. Puis il montre la complexité du processus de déploiement manuel tel qu'il est pratiqué actuellement. Enfin, après un tour d'horizon des travaux apparentés, ce chapitre conclut à la nécessité d'automatiser le déploiement des applications sur les grilles de calcul.

4.1 Déployer des applications complexes sur des grilles de calcul

4.1.1 Problématique du déploiement

La figure 4.1 illustre notre définition du déploiement d'application.

Définition 4.1 : déploiement d'application — *Le déploiement d'une application est constitué de l'ensemble des opérations qui suivent son développement dans son cycle de vie.*

Comme le montre la figure 4.1, le processus de déploiement d'une application s'étend de la description et la compilation de l'application jusqu'à son exécution effective et sa terminaison.

La description de l'application consiste à exprimer, dans un formalisme convenu, l'architecture de l'application : le ou les documents de description de l'application décrivent les parties (composants ou non) qui constituent l'application (version, localisation, *etc.*), leur assemblage, leur composition hiérarchique ou non, les interconnexions entre les constituants de l'application, la configuration de leurs paramètres, leurs dépendances vis-à-vis de fichiers de données, de bibliothèques ou DLL, ainsi que leurs dépendances par rapport à d'autres composants et leurs versions. Parallèlement à la description de l'application, les programmes qui la constituent peuvent être compilés en des fichiers exécutables, si besoin. Ensuite, l'application peut être packagée pour être distribuée.

La phase de « publication » (*release* et *advertise* en anglais) de l'application consiste à la rendre disponible pour les utilisateurs, par exemple sur un site web ou par notification dans une liste de diffusion, et éventuellement à en assurer la maintenance. La phase de « retrait » de l'application consiste à la rendre obsolète, à ne plus la diffuser ni en assurer le support.

L'installation de l'application comprend le transfert des fichiers nécessaires à son exécution (données en entrée, exécutables, dépendances[1], *etc.*) ainsi que la configuration de son environnement (variables d'environnement, permissions sur les fichiers, *etc.*). L'opération contraire est la désinstallation, qui doit éliminer les fichiers liés à l'application, et éventuellement ses fichiers de configuration. La mise à jour d'une application comprend les mêmes opérations que l'installation (transfert et configuration).

Le lancement de l'application comprend l'exécution des processus de l'application et l'activation de ses constituants (l'activation peut être précédée d'une étape de configuration dynamique de l'application et de connexion des composants le cas échéant). La terminaison de l'application consiste à arrêter ses processus pour libérer les ressources du système d'exploitation. L'adaptation dynamique de l'application consiste à modifier son comportement alors qu'elle est en cours d'exécution.

4.1.1.1 Déploiement d'application et déploiement de logiciel

D'après notre définition, le déploiement d'*application* se distingue du déploiement de *logiciel*, car il s'étend jusqu'à la phase de lancement de l'application. Traditionnellement, les problématiques du déploiement de logiciels sont, de manière non exhaustive :

- la publication et le retrait de logiciels (éventuellement à base de composants) ;
- l'installation (transfert et configuration) et la désinstallation sur un site qui demande explicitement le déploiement (ou le repliement) d'un logiciel précis ;
- la gestion des dépendances entre logiciels ou entre composants de logiciels, et l'automatisation du déploiement des dépendances si nécessaire ;
- la gestion des versions des logiciels, de leurs composants, et de leurs dépendances (cohabitation ou conflits entre différentes versions) ;
- la mise à jour et l'adaptation de logiciels ou de composants de logiciels ;

[1] Dépendances vis-à-vis de bibliothèques particulières, fichiers de configuration, *etc.*

– le partage de composants communs entre plusieurs logiciels, et la désinstallation de logiciels sans rendre les autres logiciels inopérants.

Software Dock [94, 93], ORYA [113], InstallShield [197] sous Windows, et APT ou RPM sous Linux sont des exemples d'environnements de déploiement de logiciels. Eureka [133] permet de localiser les composants dont dépend un logiciel pour les installer automatiquement dans le cadre d'une plate-forme OSGi[2]. Resolvit [144] est un outil de déploiement de services logiciels sur des plates-formes OSGi. FROGi [52, 182] est un projet qui se propose de déployer des logiciels à base de composants Fractal sur des plates-formes OSGi. Dans tous ces exemples de déploiement de logiciels, il n'est pas question de lancement d'application.

Outre les problématiques traditionnelles du déploiement de logiciels, le déploiement d'applications sur grille de calcul se distingue du déploiement de logiciels par les rôles des clients et serveurs. Dans le cadre du déploiement de logiciels, c'est le client (un terminal OSGi par exemple) qui demande à ce qu'un logiciel soit déployé sur sa plate-forme (mode « *pull* ») : les fichiers sont déplacés des serveurs de logiciels vers le client. En revanche, dans le cadre du déploiement d'applications sur des grilles de calcul, les fichiers sont déplacés du client vers les serveurs de calcul (mode « *push* »).

4.1.1.2 À l'intersection du domaine des applications et des infrastructures d'exécution

Le déploiement d'applications sur des grilles de calcul fait intervenir à la fois le domaine des applications (leur description via un ADL, leur packaging, leur cycle de vie, *etc.*) et celui des infrastructures d'exécution, avec leurs ressources distribuées et hétérogènes, leurs intergiciels d'accès, leurs politiques de sécurité, *etc.*

Diversité des types d'applications. Les grilles de calcul ne sont pas restreintes à une classe particulière d'applications : ce sont des infrastructures d'exécution génériques et nous avons besoin d'y lancer des applications de différents types, voire des applications qui mêlent plusieurs technologies, telles que les composants parallèles.

Diversité des ressources d'exécution et des moyens d'y accéder. La puissance informatique des grilles de calcul est accessible au prix d'une grande complexité d'utilisation, malgré les intergiciels qui fournissent les fonctionnalités de base pour offrir une vision unifiée et sécurisée des ressources. Cependant, il n'existe pas encore d'interface unique et normalisée pour interagir avec les intergiciels d'accès aux grilles. Ainsi, nous ne pouvons pas nous restreindre à un seul intergiciel, afin de maximiser la quantité de ressources que l'utilisateur peut impliquer dans la résolution de son problème numérique.

4.1.2 Re-déploiement d'applications

Comme l'illustre la figure 4.1, nous adoptons une vision relativement statique du déploiement d'application : nous nous intéressons au problème qui consiste à déployer une application une fois sur un ensemble de ressources fixé, et tous les déploiements d'applications sont indépendants les uns des autres. En particulier, nous ne nous focalisons pas sur les questions

[2]OSGi (*Open Services Gateway Initiative*, [214]) est une plate-forme de services JAVA pour la gestion des réseaux domestiques ; OSCAR [213] est une implémentation de OSGi.

re-déploiement, qui concernent la mise à jour ou l'adaptation dynamique de l'application, et la migration des constituants de l'application en cours d'exécution d'un ensemble de ressources vers un autre. De plus, nous nous sommes restreints aux applications statiques au paragraphe 2.3.1.2. Cependant, nous ne nions pas l'utilité du re-déploiement d'applications sur des ressources aussi dynamiques que celles des grilles de calcul.

Le re-déploiement est utile lorsque l'application a besoin de créer de nouveaux processus en cours d'exécution (`MPI_Comm_spawn` dans MPI-2 par exemple), ou bien pour faire de l'équilibrage de charge si une machine est trop chargée. Le re-déploiement peut aussi être utile lorsqu'une partie (ou l'intégralité) de l'application a été interrompue prématurément : c'est le cas si un processus de l'application est isolé en raison d'une défaillance réseau ou bien si le processus crashe (problème du système d'exploitation), c'est aussi le cas si la ressource de calcul a été allouée pour une durée de calcul qui a été dépassée (ce dernier cas est tout à fait plausible dans le cadre des grilles de calcul, où les ressources sont souvent allouées pour une durée déterminée).

Comme nous commençons l'exploration du domaine de l'automatisation du déploiement d'applications de différents types sur des grilles de calcul, nous nous restreignons dans un premier temps au déploiement statique d'applications statiques pour simplifier le problème. Toutefois, nous visons toujours un nombre significatif d'applications, car peu d'applications sont écrites pour pouvoir être migrées ou redémarrées après une défaillance (*checkpoint-restart*), et l'équilibrage de charge n'est pas forcément nécessaire si les ressources sont allouées à l'application de manière exclusive (ce qui est souvent le cas dans les grilles de calcul).

4.1.3 Discussion

Il s'agit bien du déploiement des applications métier de calcul scientifique auquel nous nous intéressons, et non du déploiement de l'infrastructure de calcul (les grilles), ni du déploiement des intergiciels d'accès aux grilles de calcul. Nous supposons que les grilles de calcul et leurs intergiciels d'accès sont déjà mis en place et opérationnels.

Sauf mention contraire, dans tout le reste du document, le terme de « déploiement » fait référence au déploiement statique sur des grilles de calcul d'applications concurrentes de calcul scientifique, statiques, et fondées sur différentes technologies de programmation. Le processus de déploiement inclut la description des applications, leur packaging, leur installation et le lancement de leur exécution.

4.2 État de l'art en matière de déploiement d'applications

Cette section présente en détails la marche à suivre pour lancer des applications distribuées (à base de composants CCM) et des applications parallèles (MPI et MVP). L'objectif est de montrer que l'état de l'art ne permet pas de se soustraire à une suite complexe d'opérations manuelles pour déployer les applications concurrentes de calcul scientifique dans l'environnement distribué des grilles de calcul.

4.2.1 Déploiement d'applications distribuées à base de composants CCM

Cette section présente la marche à suivre pour déployer une application à base de composants CORBA sur une grille de calcul gérée par Globus. Nous supposons que la phase de création des composants (définition des interfaces, développement des codes métier) a déjà été effectuée. Nous faisons également l'hypothèse que les composants sont décrits et packagés, qu'ils sont assemblés pour former l'application, et que l'assemblage est décrit et packagé. La phase de génération des descripteurs XML (`.csd` et `.cad`), de packaging et d'assemblage des composants peut se réaliser à l'aide d'outils graphiques tels que l'*Assembly and Deployment Toolkit* [167] de Frank Pilhofer, ou encore le *Packaging and Assembling Tool* d'OpenCCM[3]. En outre, ces outils permettent de configurer les valeurs initiales des attributs des composants et de définir les connexions entre les ports des instances de composants. Enfin, nous supposons que l'utilisateur possède un certificat d'authentification Globus/X.509 (chapitre 3), et que ce certificat est valide, installé, et correctement initialisé (proxy activé avec un mot de passe long).

4.2.1.1 Recherche de ressources disponibles

Dans un premier temps, l'utilisateur doit trouver des ressources disponibles pour déployer son application en interrogeant le MDS de Globus et en analysant les informations qu'il fournit en sortie :

- quels sont les ordinateurs à sa disposition ?
- quels sont leurs systèmes d'exploitation et leurs architectures matérielles ?
- quelles plates-formes sont compatibles avec quelles implémentations de composants ?
- ces ordinateurs peuvent-ils communiquer entre eux afin que puissent être établies les connexions entre les ports des instances de composants ?
- quelles sont les méthodes d'accès à ces ordinateurs distribués, tant pour y rapatrier des fichiers que pour y lancer des processus ?

4.2.1.2 Placement des composants

Ensuite, l'utilisateur doit « placer » les composants de l'application, c'est-à-dire déterminer quelle instance de composant devra s'exécuter sur quelle machine, et avec quelle implémentation des composants. Cette étape se fait en respectant les contraintes de co-localisation des instances de composants (sur une même machine, ou bien dans un même processus), la compatibilité des plates-formes de calcul disponibles avec les implémentations des composants, et les possibilités de communication entre les instances de composants qui devront être interconnectés.

4.2.1.3 Lancement du service de nommage

Puis l'utilisateur peut avoir à lancer un service de nommage (*Naming Service*) : c'est un programme CORBA auprès duquel les serveurs de conteneurs, les composants, et tout objet CORBA en général peut s'enregistrer. Le service de nommage permet de retrouver, à partir de leur nom symbolique, les références des objets CORBA qui y ont été enregistrés afin de

[3] OpenCCM est une implémentation de CCM, *cf.* section 4.3.3.3.

```
(&(resourceManagerContact="frontal.grenoble.grid5000.fr")
  (count=1)
  (executable=$(GLOBUSRUN_GASS_URL)#"/opt/x86/linux/CCM_NS.static")
  (arguments="--ior" "-")
)
```

FIG. 4.2 – Script RSL pour commander le lancement d'un service de nommage (fichier binaire local `/opt/sparc/solaris9/CCM_NS.static`) à distance via Globus.

les localiser et les contacter. La référence d'un objet CORBA, ou IOR (*Interoperable Object Reference*), permet de contacter l'objet : cette IOR est une chaîne de caractères qui encode diverses informations, telles que le type d'objet qu'elle référence, sa localisation (URL), *etc.*

Le cas échéant, l'utilisateur doit veiller à ce que ce service de nommage puisse être contacté par tous les processus CORBA qu'il va déployer par la suite. En particulier, ce service de nommage ne doit pas être isolé des autres ordinateurs par un pare-feu[4].

Le lancement du service de nommage se fait grâce au service GRAM de Globus : il faut écrire un script RSL et le soumettre (figure 4.2). Ce script RSL décrit le point de contact de la machine sur laquelle lancer le service de nommage CCM, le nombre de processus à lancer, et indique où se trouve le fichier exécutable à lancer : dans cet exemple, c'est un fichier présent localement sur le client qui sera automatiquement rapatrié sur la machine d'exécution. Pour éviter de dépendre de bibliothèques qui ne seraient pas installées, le fichier binaire est compilé statiquement.

Une fois le service de nommage lancé sur une machine adéquate, et suivant la méthode de lancement appropriée à cette ressource de calcul, l'utilisateur doit récupérer la référence CORBA de ce service de nommage (IOR).

4.2.1.4 Installation des implémentations des composants

L'utilisateur doit maintenant installer chaque implémentation de composant qu'il a sélectionnée (section 4.2.1.2) sur la machine compatible où il prévoit de lancer ce composant. Pour ce faire, il doit localiser l'implémentation de chaque composant (dans les fichiers XML de description **`.csd`**), et connaître les méthodes de transfert de fichiers possibles entre les points de stockage des implémentations et les machines qui hébergeront les composants. Pour une grille gérée par Globus, les transferts de fichiers se font par une commande telle que :

```
globus-url-copy http://component.store.com/Bank.i386.linux.static \
                gsiftp://machine.destination.fr//home/user/Bank_component
```

4.2.1.5 Lancement des serveurs de conteneur et des composants

Ensuite, l'utilisateur doit lancer les exécutables de ses composants, ou bien les serveurs de conteneurs pour le cas des composants sous la forme de DLL. Comme pour le lancement du service de nommage (section 4.2.1.3), l'utilisateur doit adapter la méthode de lancement de processus à distance en fonction de celles qui sont acceptées par chacune des ressources de

[4]Un pare-feu (*firewall*) est un dispositif informatique qui filtre les flux de communications entre les ordinateurs de différents domaines.

calcul distribuées : pour une grille gérée par Globus, l'utilisateur doit encore écrire un script RSL pour lancer ses processus à distance.

Lors de ces lancements, l'utilisateur doit passer en paramètre l'IOR du service de nommage (s'il existe) aux composants et aux serveurs de conteneur qui s'enregistrent d'eux-mêmes auprès du service de nommage. Pour les composants exécutables et les serveurs de conteneur qui ne s'enregistrent pas systématiquement auprès du service de nommage, l'utilisateur doit récupérer leurs IOR.

4.2.1.6 Instanciation et configuration des composants

Toutes les étapes précédentes sont réalisées *manuellement* par l'utilisateur, sans l'aide d'aucun outil de déploiement. Les étapes restantes peuvent s'accomplir avec l'outil de déploiement (*Assembly and Deployment Toolkit*) de Frank Pilhofer.

En analysant le fichier de description de l'assemblage de composants **`.cad`**, l'outil de déploiement sait quels sont les composants qui ne s'enregistrent pas d'eux-mêmes auprès du service de nommage : pour les enregistrer, l'utilisateur fournit à l'outil de déploiement l'IOR du service de nommage, ainsi que les IOR de ces composants exécutables non encore enregistrés (ces références ont été récupérées à l'étape précédente, section 4.2.1.5).

Puis l'utilisateur fournit à l'outil de déploiement les IOR des serveurs de conteneur, pour qu'il puisse y charger les composants sous forme de DLL (les DLL ont déjà été copiées sur les machines d'exécution à la section 4.2.1.4). À ce point, tous les composants (exécutables et DLL dans les serveurs de conteneur) ont été instanciés.

Il ne reste plus qu'à configurer les instances de composants (fixer les valeurs initiales de leurs attributs) et à les interconnecter : l'outil de déploiement accomplit cette étape automatiquement, en analysant les fichiers XML de description des composants (**`.csd`**) et de l'assemblage (**`.cad`**).

Ainsi, l'outil de déploiement de Frank Pilhofer suppose que les serveurs de conteneur et le service de nommage ont déjà été lancés, et que l'utilisateur connaît leurs références IOR. De plus, cet outil permet uniquement le déploiement de composants sous forme de DLL dans des serveurs de conteneur, mais pas le lancement de composants exécutables ou JAVA.

4.2.1.7 Discussion

Le modèle de déploiement de CCM ne spécifie pas les aspects du lancement d'applications liés aux infrastructures d'exécution. Par exemple, CCM ne précise pas comment découvrir et sélectionner les ressources de calcul, comment placer les composants sur ces ressources, comment transférer les fichiers d'implémentation des composants, ni comment lancer les processus de l'application à distance. De plus, le lancement effectif d'applications en environnement distribué reste une opération complexe, qui nécessite des compétences assez avancées en informatique. Or les utilisateurs qui veulent déployer des applications de calcul scientifique à base de composants peuvent être des physiciens, des biologistes, des chimistes, des climatologues, *etc.* dont on ne peut exiger un tel niveau d'expertise en informatique pour le lancement de leurs applications. Ainsi, il est nécessaire d'aider l'utilisateur à déployer de telles applications sans nécessiter d'expérience particulière dans le domaine des ressources informatique distribuées.

4.2.2 Déploiement d'applications parallèles

Avec l'avènement des réseaux longue distance (WAN) haute performance, les applications parallèles se sont de plus en plus développées dans les environnements de grilles de calcul. Des exemples d'applications MPI dont le ratio calcul/communication permet de les exécuter sur des grilles sont donnés dans [17], et dans [121, 68] avec une exécution sur plus de mille nœuds (calcul parallèle à gros grain, couplage de codes, *etc.*). Plusieurs implémentations de MPI sont adaptées aux grilles de calcul ou aux fédérations de clusters (figure 3.1, page 45) : MPICH-G2 [101, 205], MagPIe [107], PACX-MPI [104, 215]. Il en est de même pour les implémentations de systèmes de MVP, tels que DSM-PM2 [2] ou une version modifiée de TreadMarks [25]. Dans tous les cas, l'idée consiste à minimiser les communications sur les liens les plus lents pour masquer les latences élevées de ces liens réseau [109, 108, 100].

Cette section présente la marche à suivre pour déployer des applications parallèles (MPI et MVP) dans un environnement de grille de calcul. Ici encore, nous supposons que l'infrastructure de grille de calcul est en place, et que l'utilisateur possède un certificat d'authentification X.509 initialisé.

4.2.2.1 Applications parallèles par passage de messages MPI

Nous faisons l'hypothèse que l'utilisateur possède une ou plusieurs versions compilées de l'application, qu'il sait pour quels systèmes d'exploitation et architectures les exécutables sont destinés, qu'il sait où sont localisés les binaires et comment les récupérer (par quel protocole). L'utilisateur doit posséder cette somme de connaissances lui-même puisqu'il n'existe pas d'ADL pour décrire des applications MPI, ni de format de packaging de telles applications.

Recherche de ressources disponibles. De même que pour le déploiement d'applications CCM (section 4.2.1), l'utilisateur commence par chercher les ressources qui sont à sa disposition, en interrogeant par exemple le système d'information MDS de Globus. L'utilisateur doit se renseigner sur les systèmes d'exploitation et architectures des machines, leurs performances de calcul et nombre de processeurs, la connectivité et les performances réseau entre les machines, ainsi que les méthodes d'accès à ces ordinateurs distribués (transfert de fichiers et lancement de processus), *etc.*

Placement des processus. Toujours comme pour CCM, l'utilisateur doit placer les processus de l'application MPI. Pour ce faire, il doit connaître le nombre de processus à lancer : soit il peut le déterminer librement, soit il est contraint par l'application. C'est à l'utilisateur de connaître l'application pour savoir si elle ne fonctionne qu'avec un nombre pair de processus, qu'avec au moins 8 processus, qu'avec au plus 64 processus, *etc.*

Le placement des processus doit se faire en respectant la compatibilité entre les versions compilées disponibles de l'application et les caractéristiques des ordinateurs, en termes de systèmes d'exploitation et d'architectures. L'utilisateur doit également veiller à ce que toutes les machines sélectionnées pour l'exécution de l'application puissent communiquer directement les unes avec les autres. Si par exemple les processus sont amenés à être répartis sur plusieurs clusters, l'utilisateur doit s'assurer que le partitionnement de l'application ne nuise pas à son

```
tartopom01.irisa.fr 0 /home/user/powerPC/darwin/appli.exe
tartopom02.irisa.fr 1 /home/user/powerPC/darwin/appli.exe
parasol11.irisa.fr 1 /home/user/x86_64/linux/appli.exe
parasol12.irisa.fr 1 /home/user/x86_64/linux/appli.exe
```

FIG. 4.3 – Exemple de liste des machines et des exécutables pour lancer une application MPICH-1.

exécution : si trop de processus sont reliés par des réseaux longue distance (WAN), le temps d'exécution risque d'être fortement dégradé pour une application mal partitionnée.

Enfin, en fonction de l'implémentation de la bibliothèque MPI utilisée par les machines sélectionnées, l'utilisateur peut devoir écrire un fichier qui liste les ordinateurs et les exécutables choisis. Pour MPICH-1, ce fichier « *machinefile* » ressemble à l'exemple de la figure 4.3.

Installation des fichiers. Après avoir localisé les fichiers exécutables de l'application, les fichiers de données en entrée et les dépendances (bibliothèques, *etc.*), l'utilisateur doit les installer sur chacune des ressources sélectionnées pour l'exécution de l'application. Ces transferts de fichiers nécessitent de connaître les protocoles d'accès pour transférer les fichiers vers les ressources d'exécution.

Comme le montre la figure 4.4, la phase d'installation des fichiers peut être fusionnée avec celle de lancement des processus dans la description RSL de la tâche à soumettre à Globus : la première tâche comporte un fichier **`input.data`** à rapatrier avant exécution (**`file_stage_in`**), et l'exécutable de la troisième tâche se trouve initialement dans sur l'ordinateur de l'utilisateur (**`/local/user/my_appl.exe`**) et sera transféré vers la machine d'exécution automatiquement. En tout état de cause, ce genre de script RSL est compliqué et fastidieux à écrire pour l'utilisateur.

Lancement des processus et configuration de l'application. Une grille de calcul est le plus souvent faite d'interconnexions de réseaux hétérogènes et aux performances inégales : des implémentations telles que MPICH-G2, MagPIe et PACX-MPI savent tirer parti de la hiérarchie des performances réseau pour mettre en œuvre les opérations collectives de MPI [100, 109, 108] en minimisant les communications sur les liens les plus lents.

Avec MPICH-G2, l'information sur la topologie réseau est transmise à la bibliothèque par le biais de la variable d'environnement **`GLOBUS_LAN_ID`** : l'exemple de la figure 4.4 montre comment décrire que les ressources d'exécution de l'application MPI sont faites de trois clusters, dont deux sont situés dans un même réseau local (**`Univ_of_Stanford`**, 4 et 8 processus) et l'autre se trouve dans un autre site de calcul (**`NCSA`**, 2 processus).

Avec MagPIe et PACX-MPI, l'utilisateur doit écrire un fichier de description de la topologie réseau. La figure 4.5 donne un exemple d'un tel fichier de configuration qui décrit que l'application s'exécute sur deux clusters qui hébergent chacun quatre processus.

Discussion. Comme il n'existe pas d'ADL pour décrire des applications MPI ni de modèle de packaging, c'est l'utilisateur qui doit rassembler toutes les informations sur la nature et

```
+(&(resourceManagerContact="cluster1.site1.org")
   (count=4)
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
                (GLOBUS_LAN_ID Univ_of_Stanford)
                (LD_LIBRARY_PATH "/usr/globus/lib"))
   (file_stage_in=("http://data.store.fr/input.data" "/h/user/in.data"))
   (file_clean_up="/h/user/in.data")
   (executable="/h/user/my_MPI_app_i386"))
 (&(resourceManagerContact="cluster2.site1.org")
   (count=8)
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
                (GLOBUS_LAN_ID Univ_of_Stanford)
                (LD_LIBRARY_PATH "/ap/globus2.4.3/lib"))
   (directory="/home/john/user/"))
   (executable="/home/john/MPI_appl_sparc"))
 (&(resourceManagerContact="cluster.site2.org")
   (count=2)
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 2)
                (GLOBUS_LAN_ID NCSA)
                (LD_LIBRARY_PATH "/usr/globus2.4.3/lib"))
   (executable=$(GLOBUSRUN_GASS_URL)#"/local/user/my_appl.exe"))
```

FIG. 4.4 – Exemple de script RSL pour commander l'installation de fichiers et le lancement d'une application MPICH-G2 à distance via Globus.

```
cluster 0
process 0 1 2 3

cluster 1
process 4 5 6 7
```

FIG. 4.5 – Exemple de fichier de configuration de MagPIe qui décrit la topologie sur laquelle s'exécute une application MPI (deux clusters de quatre processus chacun).

la localisation des fichiers binaires et des dépendances (bibliothèques, fichiers de configuration, *etc.*), ainsi que sur les contraintes relatives au nombre de processus de l'application MPI (puissance de deux, nombre pair, bornes inférieure et supérieure, *etc.*).

De plus, la recherche manuelle des ressources disponibles et de leurs caractéristiques est fastidieuse. Enfin, les opérations de transfert de fichiers, de lancement des processus à distance et de configuration de l'application sont techniquement complexes et sujettes à erreurs : elles nécessitent un certain niveau d'expertise en matière de grilles de calcul.

4.2.2.2 Applications parallèles par mémoire partagée sur MVP

Pour le déploiement d'applications parallèles fondées sur la mémoire partagée par MVP, les problèmes sont exactement les mêmes que pour les applications MPI : l'utilisateur doit manuellement rechercher les ressources disponibles et leurs caractéristiques, décider du placement des processus, transférer des fichiers, et lancer les processus à distance.

Comme pour les implémentations de MPI, les MVP hiérarchiques tiennent compte de l'hétérogénéité des performances des réseaux. La hiérarchie des accès mémoire (mémoire du processeur local, d'un processeur distant dans le même cluster, ou d'un processeur distant dans un cluster distant) induit des rapports de latence entre les communications (entre les threads sur un processeur, entre les nœuds d'un cluster, ou entre les nœuds de deux clusters distincts) typiquement entre 10 et 100.

Pour la MVP DSM-PM2, la bibliothèque de communication « Madeleine3 » a besoin de connaître la topologie réseau sous-jacente. Donc une application DSM-PM2 se lance en fournissant un fichier de configuration comme celui représenté sur la figure 4.6. Ce fichier est particulièrement fastidieux à écrire, puisqu'il décrit chaque canal de communication entre chaque machine. Dans cet exemple, le fichier décrit deux clusters reliés entre eux par un réseau Ethernet/TCP, et chaque cluster comprend 2 nœuds interconnectés par SCI/SISCI : le fichier de configuration décrit les deux canaux SISCI à l'intérieur des deux clusters, et les quatre canaux TCP entre les deux paires de machines. De plus, ce fichier de configuration inclut un autre fichier qui décrit les types de réseaux (SCI/SISCI et Ethernet/TCP).

4.2.3 Discussion

Pour les types d'applications vus précédemment, nous retrouvons toujours les mêmes lacunes : l'utilisateur doit se charger de découvrir les ressources et leurs caractéristiques en interrogeant les systèmes d'information de la grille, il doit lui-même sélectionner les machines, déterminer le placement des processus ou composants, installer les fichiers (exécutables, dépendances, données) en les transférant par les protocoles supportés par les ressources sélectionnées, lancer les processus par les méthodes de soumission de tâche adaptées à chaque ressource particulière, et configurer l'application conformément à sa propre méthode de configuration (variable d'environnement, fichier de configuration, *etc.*).

Non seulement les opérations à réaliser manuellement sont techniquement complexes, mais également nous remarquons que certaines opérations sont identiques quel que soit le type d'application (découverte des ressources, placement des constituants de l'application et

```
application: {
   name: a.out;
   flavor: mad3-dsm-sisci-tcp;
   networks: {
      include: networks.cfg;
      channels: ({ net: sisci;
                   hosts: (paraski11.irisa.fr, paraski12.irisa.fr); },
                 { net: sisci;
                   hosts: (paraski13.irisa.fr, paraski14.irisa.fr); },
                 { net: tcp;
                   hosts: (paraski11.irisa.fr, paraski13.irisa.fr); },
                 { net: tcp;
                   hosts: (paraski12.irisa.fr, paraski13.irisa.fr); },
                 { net: tcp;
                   hosts: (paraski11.irisa.fr, paraski14.irisa.fr); },
                 { net: tcp;
                   hosts: (paraski12.irisa.fr, paraski14.irisa.fr); });
   };
};
```

FIG. 4.6 – Exemple de fichier de lancement d'application DSM-PM2 sur deux clusters SCI/SISCI reliés par un réseau Ethernet/TCP en utilisant la bibliothèque de communication Madeleine3.

sélection de ses versions compilées). Ainsi, l'état de l'art en matière de déploiement d'applications concurrentes de calcul scientifique ne permet pas de lancer simplement ces applications sur des grilles de calcul.

4.3 Travaux apparentés au déploiement d'applications

Cette section présente une classification des travaux qui proposent une partie des fonctionnalités nécessaires au déploiement d'applications. Nous soulignons en particulier les points qui ne permettent pas de déployer simplement des applications concurrentes de divers types (parallèles et distribuées) sur des grilles de calcul, c'est-à-dire sans nécessiter d'expertise dans les domaines techniques des infrastructures d'exécution.

4.3.1 Intergiciels d'accès aux ressources des grilles

Les intergiciels d'accès aux ressources des grilles, tels que Globus et Unicore, concernent uniquement le domaine des infrastructures d'exécution, et non celui des applications. En particulier, l'information qu'ils fournissent sur les ressources doit être filtrée et analysée par l'utilisateur. De plus, les systèmes d'information ne décrivent pas les interconnexions réseau entre les ordinateurs, alors que la topologie réseau est importante pour décider du placement des constituants d'applications distribuées. Ces intergiciels n'offrent pas de support pour le placement des constituants de l'application, ni pour la sélection de ses différentes versions compilées. Enfin, nous avons déjà montré que la soumission de tâche via Globus nécessite d'écrire un script RSL souvent complexe.

4.3.1.1 GridLab GAT / GRMS

Le *Grid Application Toolkit* (GAT [16]) du projet GridLab [15] est une plate-forme de services (portail web, sécurité, système d'information, gestion des données et des ressources) qui vise à simplifier l'utilisation des grilles de calcul par les applications.

Le GRMS (*Grid Resource Management Service*, [111]) du GAT est le service de gestion des ressources (lancement des processus sur la grille). Le GRMS joue un rôle d'ordonnanceur de tâches (section 4.3.2.1). Il gère aussi tout le processus de soumission de tâche sur des ressources de la grille à travers différents systèmes, tels que Globus, Condor, PBS, LSF, SGE, *etc.* Le GRMS peut déployer des applications séquentielles et des applications MPI sur un *unique* cluster *homogène*.

Le descripteur d'application du module GRMS (« *Job Description* ») peut seulement décrire des applications constituées d'un seul exécutable, ce qui ne convient pas pour le couplage de codes. De plus, ce descripteur d'application mêle dans un même document des informations sur l'application avec des informations sur les ressources d'exécution, alors que ces informations devraient être séparées pour pouvoir facilement déployer la même application sur d'autres infrastructures d'exécution.

4.3.1.2 Elagi

Elagi [179] est une bibliothèque d'accès aux grilles de calcul qui fournit les mêmes services que les intergiciels d'accès aux grilles, mais à un plus haut niveau d'abstraction et donc plus simplement. Par exemple, Elagi dispense d'écrire des scripts RSL. Elagi permet de lancer des processus à distance, de transférer des fichiers et d'interroger des systèmes d'information variés via Globus, Condor (un système d'ordonnancement et de soumission de tâches), et divers systèmes de batch (PBS, LSF, SGE, *etc.*, *cf.* définition 3.5, page 38).

Ainsi, Elagi offre une API plus agréable et plus simple que les intergiciels d'accès aux grilles, mais elles ne fournit aucun service en plus : en particulier, Elagi ne permet pas de sélectionner les ressources d'exécution, de placer les constituants de l'application, *etc.*

4.3.2 Ordonnanceurs de tâches

4.3.2.1 Déploiement d'applications concurrentes et ordonnancement de tâches

L'ordonnancement de tâches consiste à décider de l'allocation de processus à exécuter sur des ressources. L'ordonnancement possède une dimension temporelle puisqu'il faut déterminer dans quel ordre lancer les processus dans le temps. Typiquement, l'ordonnancement concerne des applications indépendantes, même s'il peut y avoir des contraintes sur l'ordre de leur exécution : elles ne coopèrent pas directement et ne communiquent pas entre elles. De plus, pour l'ordonnancement, les ressources sont souvent en nombre insuffisant par rapport au nombre de tâches à lancer : il y a plus de tâches que de ressources. L'objectif est donc d'utiliser les ressources le plus efficacement possible.

Pour ce qui est du déploiement d'applications concurrentes sur grille de calcul, les conditions sont différentes : les tâches à lancer appartiennent à une même application, donc elles

doivent s'exécuter simultanément pour communiquer. De plus, nous considérons que les ressources des grilles de calcul sont toujours en nombre largement supérieur aux besoins des applications. Enfin, le déploiement d'applications concurrentes sur grille de calcul comporte plus une dimension spatiale que temporelle : il faut placer une seule fois et simultanément toutes les tâches d'une application sur des ressources distribuées à sélectionner.

Ainsi, les problèmes de l'ordonnancement de tâches sont différents de ceux du déploiement d'applications concurrentes sur grille de calcul. Cependant, il est utile de les mentionner comme des travaux apparentés.

4.3.2.2 Condor-G

Le projet Condor [172] a été lancé en 1988 à l'université du Wisconsin (USA). Il avait pour objectif initial de jouer le rôle d'un système de batch distribué pour faire du calcul sur un grand nombre d'ordinateurs (*high-throughput computing*) : Condor s'interface avec PBS, LSF, SGE, *etc.* Son extension Condor-G [83, 153] repose sur Globus et a pour objectif de fédérer les ressources de calcul des ordinateurs d'une grille, afin de mettre en œuvre la « récupération de cycles » (*cf.* paragraphe 3.1.2, page 38). Condor sélectionne les ressources appropriées pour lancer une application simple (un seul programme) par le mécanisme du *MatchMaking* [140]. Le *MatchMaking* consiste à faire correspondre les contraintes des tâches à lancer avec les caractéristiques des ressources, afin d'allouer ces tâches à des machines. Les contraintes sont publiées par les tâches à lancer et par les ressources disponibles : elles concernent la nature du système d'exploitation, l'architecture des machines, la quantité de mémoire, l'espace disque, *etc.*

Cependant, Condor-G présente plusieurs limitations pour ce qui est de lancer nos applications concurrentes de calcul scientifique sur des grilles. Il alloue un *unique* programme (éventuellement parallèle) à un ou plusieurs ordinateurs. Ainsi, Condor-G n'est pas à même de lancer des applications de couplage de codes. Les seules applications concurrentes que Condor-G sait déployer sont les applications MPI, mais uniquement sur *un seul cluster homogène*. Enfin, Condor-G n'est pas en mesure de prendre en compte les contraintes des applications distribuées concernant la connectivité réseau.

4.3.2.3 APST

APST (*A Parameter Sweep Tool*, [50]) est une plate-forme de lancement d'applications paramétriques : elle lance plusieurs fois le même programme en même temps (en « parallèle ») sur plusieurs ressources de calcul avec des paramètres différents (arguments en ligne de commande ou fichiers d'entrée). Les tâches qui s'exécutent simultanément sont indépendantes et ne communiquent pas entre elles. Tout au plus les paramètres d'une vague de lancements peuvent-ils dépendre des résultats (fichiers de sortie) produits lors de la vague précédente. APST se focalise sur la co-localisation efficace des tâches qui partagent des fichiers communs de données en entrée. APST supporte plusieurs algorithmes d'ordonnancement qui se branchent comme des plugins, et repose sur Elagi pour le lancement effectif des processus dans un environnement de grille de calcul.

Les applications lancées par APST sont très simples, donc APST ne convient pas pour déployer des applications distribuées et/ou parallèles sur plusieurs ressources de calcul. De

plus, APST mélange dans un même document XML la description des tâches à exécuter avec la description des ressources, ce qui ne permet pas simplement de déployer la même application sur différentes infrastructures d'exécution.

4.3.2.4 Nimrod/G et GridWay

Le projet Nimrod [13, 208] a été lancé en 1994 à l'université de Monash, en Australie, dans le but de répartir des calculs d'applications paramétriques (calcul scientifique, simulation numérique) sur un grand nombre d'ordinateurs faisant partie d'un réseau local. Il a été étendu aux grilles de calcul par Nimrod/G [45] en utilisant la boîte à outils Globus pour la découverte des ressources. De même que APST, Nimrod/G ordonnance des tâches simples mais ne déploie pas d'applications concurrentes en milieu distribué.

Enfin, GridWay [189] est une plate-forme d'ordonnancement de tâches simples (séquentielles) sur plusieurs sites Globus. Il offre moins de fonctionnalités que Condor-G, APST et Nimrod/G.

4.3.3 Outils de déploiement spécifiques

4.3.3.1 GoDIET

DIET (*Distributed Interactive Engineering Toolbox*, [49, 176]) est un environnement de calcul de type PSE[5] pour les grilles de calcul. DIET est fondé sur l'appel de procédure à distance (RPC) et sur une organisation hiérarchique des serveurs de calcul.

GoDIET [48] est l'outil de déploiement de l'infrastructure DIET. Il est spécialisé dans le lancement de cette plate-forme exclusivement. La description de l'application (la plate-forme DIET) et la description des ressources d'exécution sont mélangées dans un unique document XML, ce qui rend difficile le déploiement de cette plate-forme sur une autre infrastructure d'exécution. De plus, il appartient à l'utilisateur de découvrir les ressources disponibles avec leurs caractéristiques, de sélectionner les machines, et de placer chacun des serveurs DIET sur les machines sélectionnées. Enfin, GoDIET est limité au protocole SSH/SCP pour transférer les fichiers de configuration de DIET et lancer les processus à distance.

4.3.3.2 GridSolve et Ninf-G

De même que DIET, NetSolve [27] est un PSE pour exécuter des applications scientifiques ; GridSolve [188] en est la version pour les grilles de calcul. NetSolve / GridSolve sont implémentés comme une bibliothèque de programmation distribuée par appel de procédure à distance (RPC), donc ils imposent un modèle de programmation et une modification du code source des programmes.

GridSolve alloue un *unique programme* (éventuellement parallèle) à un ou plusieurs serveurs d'exécution, ce qui exclut les applications de couplage de codes. De plus, GridSolve nécessite que des serveurs spécifiques soient pré-déployés par l'utilisateur avant de pouvoir

[5] Un PSE (*Problem Solving Environment*) est un environnement logiciel intégré pour la résolution d'un type particulier de problèmes de calcul numérique ; un PSE cache à l'utilisateur les détails de l'environnement logiciel et matériel des ressources.

lancer une application de calcul scientifique. Or nous ne pouvons pas faire l'hypothèse, dans un environnement de grille de calcul, que toutes les ressources hébergent un serveur spécifique pour chaque PSE, puisque les grilles de calcul ne sont pas dédiées, en général, à un type particulier d'application. Il revient à l'utilisateur de déployer les serveurs GridSolve par ses propres moyens.

Les mêmes remarques s'appliquent également au projet japonais Ninf-G [152, 209], un PSE fondé sur le RPC : l'utilisateur est responsable de déployer lui-même ses applications compilées avec la bibliothèque Ninf-G en utilisant directement Globus.

4.3.3.3 OpenCCM

OpenCCM [124, 212] est une implémentation de CCM qui est portable sur plusieurs implémentations de CORBA. OpenCCM permet de compiler, packager, assembler, déployer, installer, instancier, exécuter et gérer des applications CCM en environnement distribué. En particulier, OpenCCM interconnecte les ports de communications des composants entre eux en suivant la description de l'application CCM, et configure les paramètres des composants.

OpenCCM est restreint aux applications de type CCM, et repose sur DCI (*Distributed Computing Infrastructure*, [41]). DCI est une infrastructure distribuée que l'utilisateur doit manuellement déployer avant de lancer des applications OpenCCM. Cette infrastructure est assez lourde à déployer : l'utilisateur doit lancer manuellement sur chaque ressource distribuée un serveur web spécialisé, puis éventuellement un service de nommage CORBA, puis communiquer à chaque serveur web la référence du service de nommage (IOR). Ensuite, l'utilisateur commande à chaque serveur web le lancement d'un serveur de conteneur (*ComponentServer*) ou un gestionnaire de nœud (*NodeManager*).

De plus, l'utilisateur doit modifier la description des applications CCM qu'il veut déployer pour y faire figurer le serveur sur lequel chaque instance de composant devra être lancée (découverte et sélection manuelles des ressources, placement par l'utilisateur des composants sur les ordinateurs).

4.3.3.4 JDF

JXTA [201] est une bibliothèque de communication pair-à-pair (P2P). JDF (*JXTA Distributed Framework*, [24, 199]) est un outil de déploiement de plates-formes JXTA pour répéter des tests sous différentes configurations.

JDF est spécifique aux applications fondées sur JXTA. Il est capable de lancer des processus à distance et de transférer ses fichiers de configuration en utilisant RSH/RCP, SSH/SCP, PBS et Globus. De plus, JDF mélange dans un unique document XML la description de la plate-forme à déployer, les configuration pour les tests, ainsi que la description des ressources d'exécution sélectionnées et associées aux classes JAVA.

4.3.3.5 Pegasus

Pegasus (*Planning for Execution in Grids* [59]) est une plate-forme de déploiement d'applications sous la forme de workflows (*i.e.*, faites de tâches séquentielles) qui manipulent de

très grandes quantités de données [58]. Les dépendances entre les tâches sont uniquement liées aux flots de données entre ces tâches (pas de communication directe entre elles). Pegasus sait décrire les workflows indépendamment des ressources d'exécution (« *abstract workflow* »), pour ensuite les projeter sur les ressources disponibles des grilles, en les découvrant automatiquement. Pegasus place les tâches en fonction de la localisation des données pour produire un workflow « concret » (*concrete workflow*) : comme les données manipulées par les applications auxquelles Pegasus s'intéresse sont très volumineuses, il est souvent préférable d'installer les tâches à exécuter à proximité des données. Ensuite, le workflow concret est exécuté par Condor-G (DAGMan[6] [83]), qui lance les tâches individuelles sur les ressources indiquées par Pegasus.

Même si Pegasus se donne un objectif similaire au nôtre, à savoir masquer la complexité d'utilisation des grilles malgré les intergiciels d'accès [37], ce projet s'intéresse à un type d'applications complètement différent, faites de tâches individuelles, et il tient compte de la localisation des données. En particulier, Pegasus ne permet pas de déployer des applications concurrentes au sens de la définition 2.2 (page 10).

4.3.3.6 ProActive

ProActive (paragraphe 2.2.2.4) mêle dans un même descripteur de déploiement le placement des nœuds virtuels (par exemple des composants Fractal d'une application) avec la description des ressources (protocoles de soumission de tâches à distance et adresses IP des machines). Cette approche rend difficile le déploiement d'une même application sur différentes infrastructures d'exécution. En revanche, ProActive s'interface avec plusieurs intergiciels de soumission de tâches sur des ressources distantes (PBS, LSF, Globus, SSH, SGE, *etc.*).

4.3.3.7 ICENI

ICENI (paragraphe 2.2.2.4) offre toute une technologie de déploiement de ses composants. La sélection des ressources et le placement des composants sur ces ressources sont effectués automatiquement par ICENI en fonction de contraintes utilisateur, telles que le coût d'utilisation des ressources ou des contraintes de qualité de service [86, 85]. ICENI repose sur plusieurs intergiciels pour déployer ses composants : *fork* pour le lancement de processus localement, SSH, Globus, SGE, Condor, *etc.* [84, 164].

En revanche, ICENI ne tient pas compte de la connectivité réseau et des contraintes de communication entre les composants distribués de l'application. De plus, ICENI est limité à un type particulier de composants qui lui est propre.

4.3.3.8 SmartFrog

SmartFrog[7] [89, 231] est un environnement développé par HP Labs pour décrire, configurer, lancer et gérer des logiciels. Les logiciels supportés par SmartFrog doivent être implémentés en JAVA (ou enveloppés par un *wrapper* en JAVA).

[6] DAGMan : *Directed Acyclic Graph Manager*.
[7] SMART FR*amework for Object Groups.*

SmartFrog permet de décrire un logiciel ainsi que l'ordinateur sur lequel il doit être automatiquement installé et configuré, puis lancé. L'ensemble de ces informations (sur le logiciel et les ordinateurs cibles) est mêlé dans un seul et même document. SmartFrog sait déployer des logiciels complexes, constitués de plusieurs composants, et il orchestre leur configuration dans l'ordre approprié. Les machines sur lesquelles SmartFrog peut déployer un logiciel doivent héberger un daemon spécifique.

4.3.3.9 Concerto

Concerto [122] est une plate-forme de composants JAVA parallèles qui s'adaptent à l'évolution dynamique des ressources. Concerto impose une technologie de programmation parallèle avec des threads JAVA qui doivent s'exécuter dans la même JVM. Le modèle de communication est aussi imposé (JAVA RMI ou sockets JAVA). Concerto est focalisé sur l'adaptabilité et la connaissance de l'environnement distribué par l'application. En matière de déploiement, Concerto suppose qu'une infrastructure spécifique est déjà pré-déployée pour observer les ressources et y charger le code de l'application.

4.3.3.10 XtremWeb et BOINC

BOINC (*Berkeley Open Infrastructure for Network Computing*, [168]) est un projet de plates-formes de calcul distribué fondées sur la participation volontaire de propriétaires d'ordinateurs connectés à Internet. BOINC comprend par exemple le projet SETI@home[8] pour la recherche de vie extraterrestre par l'analyse de signaux radio.

XtremWeb [47, 235] est une plate-forme logicielle de calcul distribué fondée sur la récupération de cycles (*cf.* section 3.1.2, page 38) du même type que BOINC. XtremWeb supporte la volatilité des serveurs d'exécution et implémente des mécanismes de sécurité.

Les applications supportées par ces deux projets doivent être découpées en de nombreuses petites tâches indépendantes, donc ce modèle applicatif est très spécifique et ne correspond pas à celui du couplage de codes en particulier, mais plutôt aux applications paramétriques par exemple. XtremWeb implémente un ordonnanceur simple de tâches qui peuvent être concurrentes : il sait lancer des applications RPC ou MPI. Cependant, les clients ne peuvent pas exécuter d'applications arbitraires : ce sont les administrateurs de la plate-forme XtremWeb qui choisissent et insèrent les applications supportées. Le client ne peut pas lancer d'autres applications, il peut simplement changer les fichiers de données en entrée et les paramètres de calcul.

Ces deux projets consistent à faire calculer des ordinateurs distribués à travers Internet au lieu d'être inactifs, par le biais d'économiseurs d'écran par exemple. Pour que l'application progresse, il faut donc attendre que des ressources se libèrent. Le client qui a un calcul à exécuter ne choisit pas quand son calcul est lancé et s'exécute. Les serveurs répartis sur Internet (« *workers* ») viennent chercher des tâches à exécuter lorsqu'ils sont inactifs : ce sont les serveurs d'exécution qui prennent l'initiative de demander qu'une tâche leur soit allouée (mode « *pull* »). Le déploiement d'une plate-forme XtremWeb ou BOINC a lieu dynamiquement à l'initiative des serveurs de calcul : le client qui a une application à exécuter n'est pas à l'origine du déploiement de la plate-forme.

[8] *Search for ExtraTerrestrial Intelligence*, `http://setiweb.ssl.berkeley.edu/`.

Ainsi, l'objectif de ces projets est différent de notre problème. Dans notre optique, c'est le client qui choisit son application concurrente de calcul scientifique arbitrairement, et qui prend l'initiative de la lancer sur des ressources dédiées au calcul en mode « *push* ». Notre optique de travail impose donc en particulier de découvrir des ressources, et de placer les constituants des applications sur les ressources d'exécution sélectionnées, tandis que ces étapes n'apparaissent pas dans XtremWeb ou BOINC.

4.3.3.11 JAVA WebStart

JAVA WebStart [99] est une plate-forme interactive d'installation sécurisée de logiciels et de lancement d'applications JAVA simples à partir d'un navigateur web. JAVA WebStart s'assure que la version téléchargée du logiciel est toujours la plus récente. Il s'agit d'un *lanceur d'application*, et non d'un outil de déploiement : il fonctionne en mode « *pull* » (section 4.1.1.1) et ne permet donc pas de déployer des applications *distribuées* sur des ressources distantes.

4.3.4 Travaux de modélisation

4.3.4.1 Workflow de déploiement d'applications à base de composants

A. Flissi et Ph. Merle (LIFL, France) proposent de modéliser l'exécution des tâches de déploiement d'applications à base de composants sous la forme d'un workflow [72, 73], *i.e.* un graphe de tâches élémentaires : installation et instanciation des composants, connexion de leurs ports, configurations de leurs paramètres, activation. Certaines tâches sont dépendantes d'autres tâches qui doivent avoir été exécutées auparavant, tandis que d'autres tâches peuvent s'exécuter en parallèle. L'exécution du workflow est orchestrée automatiquement en fonction des dépendances entre les tâches. Ce travail vise à être indépendant des technologies de composants (CCM, Fractal, *etc.*), mais il reste limité aux applications distribuées à base de composants.

De plus, les activités de déploiement en amont de l'installation ne sont pas prises en compte : découverte et sélection des ressources, placement des composants, sélection des implémentations des composants, *etc.* Enfin, la plate-forme technologique spécifique sur laquelle les composants sont déployés est supposée déjà mise en place par l'utilisateur (manuellement).

4.3.4.2 MDA D&C

La spécification du déploiement et de la configuration des applications dirigée par les modèles (MDA D&C[9], [130]) de l'OMG définit un algorithme générique de déploiement d'applications à base de composants. Le processus de déploiement comporte les étapes suivantes :

1. packaging, publication ;
2. planification du déploiement, afin de déterminer sur quelle ressource chaque composant s'exécutera, et sélectionner une implémentation pour chaque instance de composant ;
3. préparation : transfert des fichiers binaires vers les machines sélectionnées ;

[9] *Model-Driven Architecture, Deployment & Configuration.*

4. lancement : instanciation des composants, connexion de leurs ports, configuration de leurs paramètres, démarrage.

L'une des originalités de MDA D&C est de donner une liste des exceptions pour décrire les erreurs qui peuvent survenir lors du processus de déploiement.

Cette spécification propose une vision de très haut niveau, et n'entre pas dans les détails pour expliquer comment se réalisent en pratique les différentes étapes du processus de déploiement. En particulier, elle reste indépendante des infrastructures d'exécution, et ne précise pas comment les processus peuvent être lancés dans un environnement distribué.

4.3.4.3 JSDL

JSDL (*Job Submission Description Language*, [21]) est une spécification du GGF en cours d'élaboration. Elle définit la structure et la sémantique d'un langage au format XML pour décrire les ressources dont une application a besoin pour son déploiement. JSDL inclut une description très sommaire de l'application à déployer ainsi que la description des ressources sur lesquelles les processus de l'application doivent être lancés. C'est de manière optionnelle seulement qu'un exécutable peut être associé à une ressource. Parmi les idées intéressantes de JSDL, nous notons la possibilité de spécifier si une partie d'application (un processus ou autre) doit avoir un accès *exclusif* à une ressource. D'ailleurs, cette idée pourrait aussi être utile dans la description des ressources : il se peut qu'une ressource (un réseau Myrinet par exemple) ne puisse être utilisée que par une seule application à la fois, un seul processus, ou bien un seul utilisateur, un seul groupe UNIX, *etc.*

Un inconvénient majeur de JSDL est de mélanger la description des ressources avec la description de l'application. La description d'application par JSDL est simpliste (l'application est faite d'exécutables indépendants), partielle (la structure de l'application n'est pas décrite), et donc insuffisante pour pouvoir décider de son placement sur les ressources en connaissance de cause. La description des contraintes sur les ressources, quant à elle, est intéressante : JSDL permet de décrire des besoins précis imposés par l'application sur les ressources. Cependant, le placement de l'application sur les ressources est facultatif pour JSDL : le programme qui consomme un tel document doit encore faire des choix, trouver des ressources, *etc.* Enfin, JSDL ne spécifie pas comment, en pratique, les processus peuvent être lancés sur les ressources.

4.3.4.4 CDDLM/GGF et SDD/OASIS

CDDLM (*Configuration Description, Deployment, and Life-cycle Management*, [170]) est un groupe de travail du GGF qui s'intéresse à la description de configuration, au déploiement et à la gestion de services (web services). CDDLM s'apparente plus au déploiement de logiciel que d'application : le travail de modélisation et de spécification de CDDLM est largement dérivé du système SmartFrog de HP. Par exemple, CDDLM [34] ne s'occupe pas de placer les services à déployer : la spécification suppose que les ressources ont déjà été allouées. De plus, CDDLM prévoit de déployer des services sur plusieurs ordinateurs distribués, mais il s'agit alors d'un seul et même service qui est répliqué à l'identique : les instances du service déployé sont indépendantes les unes des autres, et ne communiquent pas entre elles.

Initié en avril 2005, SDD (*Solution Deployment Descriptor*, [227]) est un projet du consortium OASIS[10]. SDD n'a pas encore produit de résultats, mais vise à décrire des logiciels variés sous la forme de packages, dans le but de les installer sur des plates-formes hétérogènes. Cette description devra être indépendante de la technologie d'installation (RPM, InstallShield, *etc.*). SDD vise également la gestion du cycle de vie du logiciel à déployer. Les travaux de SDD sont très proches de ceux du groupe de travail CDDLM : il s'agit de déploiement de logiciels et non d'applications (*cf.* section 4.1.1.1), et il n'est pas question de sélection des ressources sur lesquelles déployer les logiciels.

4.3.5 Discussion

Cette section a présenté des travaux apparentés au déploiement d'applications sur des grilles de calcul. L'abondance de ces travaux met en évidence l'importance que la communauté des grilles accorde aux questions de déploiement. L'intérêt suscité par la nécessité de simplifier le déploiement d'applications est encore démontré par la constitution récente du groupe de travail RSS du GGF. Le RSS-WG (*Resource Selection Services Working Group*, [225, 57]) vise à définir les interfaces des services de sélection des ressources d'exécution pour une application, et à spécifier les protocoles d'interaction entre ces services et les autres services (soumission de tâches, systèmes d'information sur les ressources, réservation de ressources, *etc.*). Le RSS-WG va se focaliser en particulier sur

- la sélection de ressources *candidates* qui *sont susceptibles* d'héberger une application (CSG, *Candidate Set Generator*),
- et la sélection des ressources sur lesquelles l'application *s'exécutera effectivement* (EPS, *Execution Planning Service*).

La section 4.2 a présenté les difficultés à déployer les applications concurrentes de calcul scientifique sur des grilles de calcul. L'analyse de ces difficultés nous conduit à attendre les propriétés suivantes pour le déploiement d'applications :

- autoriser plusieurs types d'applications choisis arbitrairement par l'utilisateur ;
- avoir une description de l'application qui soit complètement indépendante des ressources d'exécution, afin de permettre facilement son déploiement sur des infrastructures différentes ;
- autoriser un certain contrôle de l'utilisateur sur tout le processus de déploiement, par exemple si l'utilisateur souhaite que son application ne soit pas exécutée dans tel site de calcul pour des raisons de confiance, *etc.* ;
- découvrir automatiquement les ressources disponibles ;
- prendre en compte les contraintes de connectivité réseau entre les constituants des applications, ainsi que les caractéristiques des connexions entre les ressources ;
- filtrer et sélectionner automatiquement les ressources pour l'exécution ;
- placer automatiquement les constituants des applications sur les ressources en fonction des contraintes applicatives et des caractéristiques des ressources ;
- transférer automatiquement les fichiers (exécutables, dépendances, données) vers les ressources choisies pour l'exécution des applications ;
- lancer automatiquement les processus à distance en s'interfaçant avec plusieurs intergiciels de soumission de tâche des grilles de calcul ;

[10] *Organization for the Advancement of Structured Information Standards, [210].*

– configurer l'application automatiquement, en lui fournissant toutes les informations dont elle a besoin sur son environnement (variables d'environnement, fichiers de configuration, *etc.*) ;
– et enfin, ne pas nécessiter de daemon spécifique à un type particulier d'application qui soit en attente sur chaque ressource de la grille (à usage générique).

Le tableau 4.1 récapitule les propriétés que satisfont les différents travaux apparentés au déploiement d'applications concurrentes sur des grilles de calcul. Il en ressort qu'aucun des travaux à notre connaissance ne satisfait pleinement l'intégralité des propriétés que nous souhaitons réunir pour le déploiement automatique d'applications de calcul scientifique en environnement de grilles.

4.4 Conclusion

Ce chapitre a commencé par définir ce que nous entendons par déploiement d'applications concurrentes de calcul scientifique sur des grilles de calcul. En particulier, cette activité ne se limite pas à l'installation de logiciels, mais elle va jusqu'au lancement et à l'exécution effective de l'application. Dans un premier temps, nous nous restreignons à une vision statique du déploiement pour explorer le domaine. L'une des originalités de notre objectif est de s'interfacer à la fois avec le monde des applications (et leurs technologies de déploiement) et avec celui des infrastructures d'exécution à usage générique, c'est-à-dire non dédiées à un type d'application bien précis.

Ensuite, nous avons illustré la difficulté actuelle à déployer des applications concurrentes sur des grilles de calcul. Les marches à suivre sont techniquement complexes et manuelles : l'utilisateur doit maîtriser les modèles de déploiement des applications ainsi que les différentes infrastructures d'exécution et leurs intergiciels d'accès. De plus, nous avons remarqué que certaines opérations du processus de déploiement sont communes, quel que soit le type d'application à lancer. Il est donc souhaitable de factoriser ces opérations.

Enfin, bien que l'état de l'art en matière de déploiement soit très riche, il ne permet pas actuellement aux physiciens, chimistes, biologistes, géologues, *etc.* (non nécessairement experts en informatique) de déployer *simplement* leurs applications scientifiques sur des grilles, qui offrent pourtant la puissance de calcul dont ils ont besoin.

Aussi est-il indispensable de faciliter l'activité de déploiement. La deuxième partie de ce document décrit une manière de simplifier le processus de déploiement en l'automatisant avec un outil intégré.

	plusieurs types d'applications	description d'application indépendante des ressources	sélection des implémentations de l'application	contrôle de l'utilisateur	découverte des ressources	interconnexion réseau	sélection des ressources	placement des constituants de l'application	transfert des fichiers (y compris les dépendances)	lancement de processus via plusieurs intergiciels	configuration de l'application	absence de daemon spécifique
Condor-G	P	O	O	P	O	N	O	P	N	O	N	O
ICENI	N	O	O	N	P	N	O	O	N	O	N	O
Elagi	O	N	N	O	O	N	N	N	O	O	P	O
GridLab GAT/GRMS	P	N	N	N	P	N	O	O	O	O	N	O
Pegasus	N	O	N	P	O	N	O	O	N	P	N	O
Globus/Unicore	O	N	N	O	P	N	N	N	O	P	P	O
Nimrod/G	N	N	N	P	O	N	O	O	P	N	N	O
JDF	N	N	N	O	N	N	N	N	O	O	O	O
APST	N	N	N	N	N	N	O	P	O	O	N	O
XtremWeb	N	N	N	P	N	N	N	P	O	O	N	N
MDA D&C	N	O	P	N	N	N	P	P	P	N	O	–
Concerto	N	O	N	N	O	N	O	N	N	N	P	N
GridSolve et Ninf-G	N	N	N	P	P	N	O	O	P	N	N	N
GoDIET	N	N	N	N	N	N	N	N	O	N	O	O
ProActive	P	N	N	P	N	N	N	N	N	O	P	N
SmartFrog	N	N	N	P	N	N	N	N	O	N	O	N
CDDLM	N	N	N	N	N	N	N	N	O	N	O	–
OpenCCM/workflow	N	N	N	N	N	N	N	N	O	N	O	N

TAB. 4.1 – Récapitulatif des propriétés des travaux apparentés au déploiement d'applications concurrentes sur des grilles de calcul. O : propriété satisfaite ; N : propriété non satisfaite ; P : propriété partiellement satisfaite ; – : propriété non évoquée.

Deuxième partie

Automatisation du déploiement d'applications sur des grilles de calcul

Chapitre 5

Modèle de déploiement automatique d'applications sur des grilles de calcul

Sommaire

La première partie de ce document a présenté les applications de calcul scientifique qui nous intéressent : elles sont de types variés (distribuées, parallèles), voire mixtes (composants parallèles). Nous avons également présenté les grilles de calcul, qui sont des infrastructures d'exécution distribuées à même de fournir la puissance de calcul dont les applications ont besoin. Cependant, la complexité des applications et des grilles de calcul rend le déploiement de ces applications particulièrement ardu, et exige des compétences importantes dans les domaines des technologies de lancement d'applications et d'accès aux ressources hétérogènes des grilles via différents intergiciels.

L'objectif de ce chapitre est de présenter notre approche pour simplifier le déploiement d'applications concurrentes de calcul scientifique sur des grilles de calcul. Ce chapitre commence par décrire l'architecture générale de notre modèle de déploiement automatique en justifiant nos choix. Puis il détaille chaque point particulier de l'architecture générale en expliquant son rôle. Enfin, ce chapitre conclut sur les avantages et les limitations de notre approche.

5.1 Notre modèle de déploiement automatique

5.1.1 Simplifier en automatisant

Pour résumer, nous avons besoin de simplifier le déploiement d'applications de types variés (parallèles et distribuées), voire mixtes (composants parallèles) sur des grilles de calcul. Nous faisons l'hypothèse que les applications sont déjà développées, compilées, et assemblées s'il s'agit de composants. Cette hypothèse est raisonnable, car le déploiement se situe après la phase de développement. Pour simplifier le début de l'exploration du domaine, nous nous intéressons au déploiement d'applications statiques (*cf.* section 2.3.1.2, page 22).

En ce qui concerne les grilles de calcul, nous supposons que leurs ressources sont toutes accessibles via un intergiciel d'accès, tel que Globus ou UNICORE. Nous faisons également l'hypothèse que l'utilisateur a déjà acquis les droits d'accès aux ressources de la grille (certificat Globus/X.509 valide et initialisé, par exemple).

5.1.1.1 Automatisation

Comme l'a montré le chapitre 4, le déploiement des applications de calcul scientifique sur des grilles de calcul est une opération complexe, que l'état de l'art ne parvient pas à simplifier de manière satisfaisante. Aussi est-il indispensable de faciliter le processus de déploiement d'applications sur des grilles, notamment pour les scientifiques qui ne sont pas experts en informatique des grilles de calcul.

Pour *simplifier* le déploiement, nous choisissons de l'*automatiser* sans modification des applications (leur code et leur structure restent inchangés), et sans modification des intergiciels d'accès aux grilles de calcul. En effet, nous ne voulons pas imposer aux développeurs d'adapter les applications pour le déploiement ou pour l'infrastructure d'exécution particulière qu'est une grille de calcul : l'effort de redéveloppement d'une application ne va pas dans le sens de la simplification de la tâche de l'utilisateur qui veut lancer son calcul numérique.

De plus, pour favoriser l'acceptation de nos résultats, nous choisissons de fonder nos travaux sur les intergiciels d'accès aux grilles tels qu'ils existent déjà, et tels qu'ils sont mis en place sur les ressources. Donc plutôt que de soumettre directement le déploiement d'une application à un intergiciel d'accès aux grilles, nous proposons une couche logicielle supplémentaire entre l'application de l'utilisateur et les intergiciels de grilles. Cette solution permet de laisser toute leur généricité aux intergiciels, et de n'imposer aucune modification de ces intergiciels d'accès aux grilles.

Cette couche logicielle d'automatisation du déploiement masque la complexité des applications et des grilles de calcul en prenant en charge les opérations manuelles de l'utilisateur

mentionnées dans le chapitre 4 (section 4.2). Ce travail va donc dans le sens de l'un des objectifs initiaux des grilles, à savoir la transparence d'utilisation de la puissance de calcul analogue à la transparence d'accès à la puissance électrique.

Enfin, cette couche logicielle permet de factoriser les opérations communes au déploiement d'applications de différents types, telles que la découverte des ressources en interrogeant les systèmes d'information de la grille, la sélection des machines d'exécution, le placement des processus, *etc.*

5.1.1.2 Outil intégré

Nous présentons notre solution de déploiement d'applications comme un outil intégré plutôt qu'une bibliothèque. Le choix de la forme d'un outil intégré va dans le sens de la simplification d'utilisation : il permet à des utilisateurs non experts en informatique de déployer leurs applications de calcul scientifique grâce à un programme interactif. Nous nous sommes refusés à faciliter le déploiement par le biais d'une bibliothèque de programmation pour ne pas imposer à l'utilisateur d'écrire du code supplémentaire pour réaliser le déploiement.

Cet outil intégré peut servir de moteur derrière un portail web qui propose le déploiement automatique d'applications. Dans ce cas, le portail web est un habillage possible du même service de déploiement automatique.

5.1.2 Architecture générale

Cette section présente l'architecture générale que nous proposons pour le processus de déploiement automatique d'applications de types variés (voire mixtes) sur des grilles de calcul. Puis nous énumérons les tâches élémentaires que nous avons identifiées avant d'analyser la modularité et la flexibilité de cette architecture.

5.1.2.1 Enchaînement des opérations élémentaires

La figure 5.1 illustre notre architecture du processus de déploiement automatique d'applications. En entrée, l'outil de déploiement a besoin de la description de l'application à déployer, ainsi que la description des ressources disponibles de la grille de calcul (section 5.2). L'outil de déploiement peut également accepter une description de paramètres de contrôle (section 5.2.3), qui permettent à l'utilisateur de conserver un certain contrôle sur le processus de déploiement. Par exemple, l'utilisateur peut exiger qu'aucune machine sous Linux ne soit utilisée, ou bien que les ressources sélectionnées ne soient pas payantes, ou encore demander que les ressources d'exécution soient les plus puissantes possible.

Dans cette architecture, la description *spécifique* d'une application est donnée dans un format qui dépend du type de l'application : les applications CCM sont décrites par des descripteurs XML `.cad` (*cf.* section 2.2.2.5, page 16), les applications parallèles MPI sont décrites en utilisant un autre formalisme (*cf.* section 7.1, page 119), *etc.* Chaque description *spécifique* d'application est convertie en une description *générique*. Le format de description générique d'application est unique, quel que soit le type spécifique d'application à déployer. Le mécanisme de conversion et le modèle de description générique d'application sont présentés dans le chapitre 8 (section 8.1, page 138).

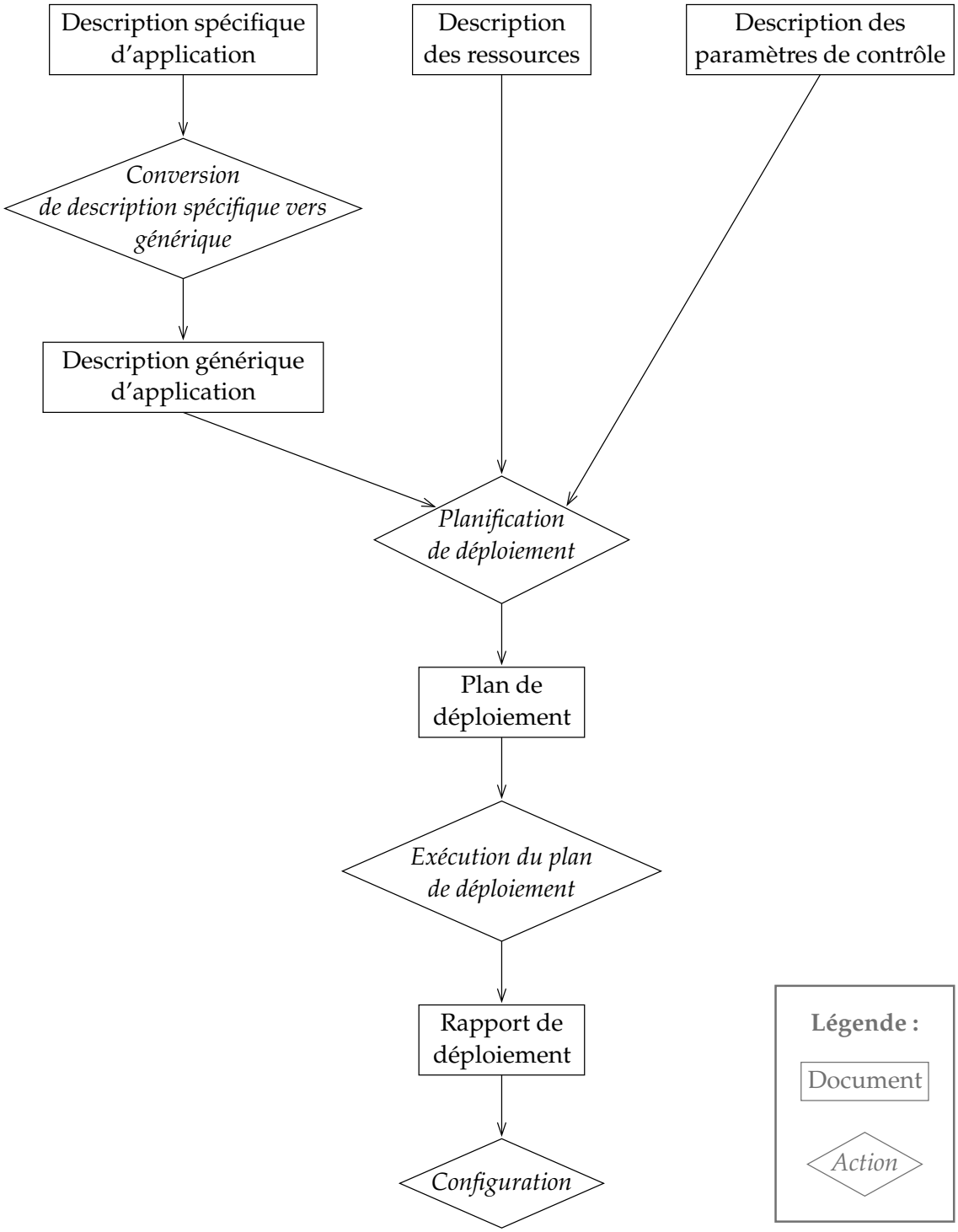


FIG. 5.1 – Architecture générale du déploiement automatique d'applications de différents types.

Le planificateur de déploiement se trouve au cœur de l'outil de déploiement automatique. Il accepte en entrée la description générique de l'application, la description des ressources de la grille, et éventuellement des paramètres de contrôle. Le planificateur de déploiement possède les responsabilités suivantes :

- sélectionner les ressources (de calcul, de communication, de stockage, *etc.*) pour exécuter l'application ;
- pour les ressources d'exécution qui possèdent plusieurs protocoles de lancement de processus (ou méthodes de soumission de tâches), sélectionner l'un des protocoles ;
- placer les constituants de l'application (processus, composants, *etc.*) sur les ressources choisies ;
- sélectionner les implémentations des constituants de l'application (par exemple, un composant ou un programme peut être compilé pour différents systèmes d'exploitation et architectures matérielles).

Le planificateur de déploiement, qui est présenté dans la section suivante, produit en sortie un « plan de déploiement », qui spécifie tous les choix qui ont été faits lors de la phase de planification.

Ensuite, le plan de déploiement est exécuté, comme le détaille la section 5.4, pour effectivement lancer l'application (après avoir mis en place son environnement d'exécution) sur les ressources choisies et avec les implémentations sélectionnées. L'exécution du plan de déploiement produit un rapport de déploiement. Pour chaque instance de programme lancée sur chaque machine, le rapport de déploiement indique si le lancement a eu lieu avec succès ou bien s'il a échoué ; il précise également quelles sont les coordonnées des constituants de l'application effectivement lancés, telles que les PID[1] des processus, les numéros de tâches soumises via Globus, les références d'objets (IOR) pour les applications CCM, *etc.*

Enfin, l'application peut être configurée, grâce aux coordonnées des constituants de l'application effectivement lancés qui se trouvent dans le rapport de déploiement. La phase de configuration correspond par exemple, pour une application CCM, à interconnecter les composants qui doivent l'être, à fixer les valeurs initiales de leurs paramètres (jusqu'à l'appel de la méthode **`configuration_complete()`**), et à activer l'application (par appel de la méthode **`run`** de l'ORB[2]).

5.1.2.2 Planification du déploiement

Le planificateur de déploiement prend toutes les décisions concernant le déploiement de l'application sur la grille : toutes ces décisions sont consignées dans le plan de déploiement que le planificateur produit en sortie. Ainsi, l'exécution du plan de déploiement qui vient ensuite ne fait intervenir aucun choix : les machines et méthodes de soumission de processus à distance sont sélectionnées, les constituants de l'application sont placés sur les ressources, et les implémentations des programmes de l'application sont déterminées.

Comme toute l'intelligence de l'outil de déploiement réside dans cette phase de planification, le planificateur est difficile à concevoir et à mettre en œuvre : il doit tenir compte des besoins exprimés dans la description générique des applications, il doit prendre en compte les contraintes imposées par les ressources disponibles, et il doit satisfaire les exigences que l'utilisateur exprime au travers des paramètres de contrôle. C'est la raison pour laquelle nous

[1] PID : *Process IDentifier*, numéro de processus.
[2] ORB : *Object Request Broker*, le « moteur » d'exécution d'une application CORBA.

voulons minimiser le nombre d'implémentations de planificateurs de déploiement. Pour ce faire, nous avons introduit la notion de description générique d'application, qui sert d'interface unique entre la description de l'application et le planificateur. Ainsi, un seul planificateur est capable de placer les constituants de n'importe quel type d'application dont la description spécifique aura été convertie dans le format de description générique d'application.

Planifier n'est pas ordonnancer, planifier n'est pas *seulement* placer. À la lumière de la section 4.3.2.1 (page 64), il apparaît que la planification de déploiement n'est pas de l'*ordonnancement*. Même si le plan de déploiement peut spécifier un ordre d'exécution pour les opérations de déploiement, l'objectif de la planification n'est pas d'allouer un maximum de processus à des ressources de calcul pour qu'ils s'exécutent en un minimum de temps. De plus, parmi les hypothèses fréquentes de l'ordonnancement [39], nous relevons l'indépendance entre les tâches (même si les sous-tâches peuvent avoir des dépendances au sein d'une tâche qui s'exécute sur une machine), ainsi que la connaissance des temps d'exécution de chaque tâche sur chaque machine. En ce qui concerne la planification, la connaissance des temps d'exécution n'est, en général, pas nécessaire, puisque la dimension temporelle est absente de la planification pour les applications statiques auxquelles nous nous intéressons : l'objectif est d'exécuter une seule application, et non un ensemble de tâches.

De plus, la planification ne se limite pas au *placement* des constituants de l'application sur des ressources. Le planificateur est en outre responsable de la *sélection* des ressources, des protocoles de lancement de processus à distance, ainsi que des implémentations des programmes de l'application.

5.1.2.3 Modularité du modèle de déploiement automatique

L'architecture de déploiement automatique que nous proposons est modulaire :

- elle supporte explicitement plusieurs types d'applications ;
- elle permet d'adopter différentes stratégies de planification, en laissant l'utilisateur sélectionner un algorithme de planification particulier par le biais des paramètres de contrôle ;
- ce modèle n'est pas lié à un intergiciel particulier pour accéder aux ressources des grilles, puisqu'il est fondé sur les fonctionnalités élémentaires
  - de découverte des ressources,
  - de lancement de processus à distance,
  - et de transferts de fichiers.

  Comme ces trois opérations sont indépendantes les unes des autres, un intergiciel peut être utilisé pour lancer une tâche et un autre intergiciel pour une autre tâche ; ou bien un intergiciel peut être utilisé pour se renseigner sur une ressource via un système d'information, et un autre intergiciel pour y soumettre une tâche à exécuter.

C'est la séparation nette des différentes phases du déploiement (figure 5.1) qui autorise cette modularité.

5.1.2.4 Flexibilité du déploiement

L'architecture de déploiement automatique que nous proposons est flexible :

– il existe plusieurs points d'entrée dans le processus de déploiement automatique : par exemple, l'utilisateur peut fournir au planificateur de déploiement une description générique d'application plutôt que sa description spécifique ; ou bien il peut directement fournir un plan de déploiement à exécuter ;
– il existe plusieurs points de sortie : par exemple, l'utilisateur peut interrompre le processus après obtention du plan de déploiement, contrôler et éventuellement modifier manuellement ce plan, puis réinjecter le plan de déploiement pour l'exécuter ;
– les paramètres de contrôle fournis par l'utilisateur lui permettent de rester maître du processus automatique de déploiement, à condition que le planificateur soit assez puissant pour pouvoir respecter toutes les contraintes exprimées par les paramètres de contrôle.

5.1.3 Discussion

On retrouve bien dans notre architecture générale du déploiement automatique toutes les étapes manuelles décrites dans la section 4.2 (page 55).

Un avantage important de ce modèle est de préserver l'indépendance entre :
– la description de l'application ;
– la description des ressources de l'infrastructure d'exécution ;
– et les exigences de l'utilisateur pour une exécution particulière de l'application.

Cette indépendance permet de :
– déployer une même application dans différents environnements d'exécution (grilles ou autre) sans changer la description de l'application ;
– déployer différentes applications sur une même infrastructure d'exécution sans modifier la description des ressources ;
– personnaliser chaque exécution d'une même application sur une même grille en ne modifiant que les paramètres de contrôle.

Parmi les points que nous avons choisi d'écarter pour commencer l'exploration du domaine de l'automatisation du déploiement d'applications, notre modèle d'architecture ne supporte pas le déploiement d'applications dynamiques. En effet, notre modèle n'apporte pas de solution aux applications qui, une fois déployées, prennent l'initiative de lancer de nouveaux processus pour s'exécuter (*cf.* section 2.3.1.2, page 22).

Enfin, notre architecture de déploiement ne gère pas l'intégralité du cycle de vie des applications. Le modèle s'arrête après le lancement de l'exécution, la configuration et l'activation de l'application (voir les étapes encadrées en gras sur la figure 4.1, page 52). Cependant, les coordonnées incluses dans le rapport de déploiement permettent de surveiller l'exécution de l'application (interroger son statut : suspendue, terminée avec succès, en cours d'exécution, avortée, *etc.*), et la contrôler (la suspendre, la relancer, l'interrompre prématurément, *etc.*).

5.2 Informations nécessaires en entrée

De la précision des informations fournies en entrée à l'outil de déploiement dépend la pertinence des choix faits par le planificateur, et donc la qualité du déploiement automatique. Cette section présente les trois informations dont l'outil de déploiement a besoin pour lancer une application sur une grille : la description de l'application, la description des ressources disponibles, et la description des paramètres de contrôle.

5.2.1 Description de l'application à déployer

Chaque application se décrit en utilisant un formalisme qui lui est propre : les applications CCM sont faites de composants configurables et interconnectés, les applications MPI sont constituées de processus, *etc.* Nous appelons « description spécifique d'application » la description d'une application exprimée dans un formalisme qui est propre au type particulier de l'application. Par exemple, la description spécifique d'une application CCM est exprimée dans un fichier `.cad`.

En général, les informations qui peuvent être présentes pour décrire une application sont, de manière non exhaustive :

- la liste des programmes qui constituent l'application ;
- la localisation des implémentations (versions compilées) de ces programmes ;
- la nature et la version des systèmes d'exploitation et des architectures matérielles pour lesquels les implémentations disponibles ont été compilées ;
- le nombre d'instances qui devront être déployées pour chaque programme ;
- des contraintes de co-localisation entre les instances de programmes à lancer ;
- les interconnexions entre les instances de programmes qui pourraient être amenées à communiquer durant l'exécution de l'application ;
- les dépendances des programmes vis-à-vis de bibliothèques, fichiers de configuration ou de données, environnement d'exécution, *etc.* dont la version peut être précisée ;
- les paramètres de configuration et l'environnement (variables, répertoire courant, *etc.*) des instances de programmes ;
- les arguments à passer en ligne de commande aux instances de programmes.

Certaines de ces informations seront utiles pour la planification du déploiement : elles seront alors traduites dans la description générique d'application comme le montre la section 8.1 (page 138). D'autres informations seront utiles pour l'exécution du plan de déploiement et la configuration de l'application après lancement (*cf.* section 5.4).

Comme l'a mis en évidence le chapitre 2, tous les types d'applications ne possèdent pas de modèle de description. Par exemple, il n'existe pas de formalisme pour décrire une application MPI : la section 7.1 (page 119) présente notre solution pour combler cette lacune.

Enfin, l'outil de déploiement doit pouvoir accepter aussi bien un descripteur spécifique d'application qu'une application packagée. Un package d'application contient au moins un descripteur de l'application, et éventuellement les exécutables des programmes qui la composent, ses dépendances, *etc.*

5.2.2 Description des ressources de la grille de calcul

L'outil de déploiement doit connaître les ressources disponibles de la grille de calcul. Pour ce faire, il doit être capable de découvrir ces ressources par plusieurs moyens et *via différents intergiciels*, puisque tous les sites d'une grille ne sont pas forcément gérés par un seul et même intergiciel. Par exemple, une partie des ressources peut être décrite dans un fichier téléchargeable par HTTP, et une autre partie découverte par le MDS de Globus.

L'information sur les ressources porte sur les *nœuds de calcul et de stockage* : nombre et vitesses des processeurs, système d'exploitation, architecture matérielle, taille mémoire, espace disque et points de montage des systèmes de fichiers, *etc.* En plus de ces informations sur les nœuds, les informations sur les *liens réseaux* entre les nœuds sont aussi cruciales pour la

planification, puisque les constituants distribués d'une application concurrente peuvent être amenés à communiquer, et l'utilisateur peut avoir des exigences précises sur les capacités de communication entre les parties d'une application. Les informations sur les réseaux de communication concernent non seulement les caractéristiques numériques de performance des liens réseau (latence, débit, taux de perte, gigue, *etc.*), mais également la topologie d'interconnexion réseau entre les nœuds de calcul et de stockage. Nous présentons dans le chapitre 6 un modèle de description de la topologie réseau qui sait décrire des pare-feux, des technologies réseau non IP, *etc.*

De plus, la description des ressources de calcul, de stockage et des liens réseau contient des informations *statiques* (architecture matérielle, nature des systèmes de fichiers, topologie réseau), mais des données *dynamiques* doivent pouvoir aussi être obtenues par l'outil de déploiement, telles que la charge instantanée d'un processeur, l'espace disque disponible sur un point de montage, le débit instantané sur un lien réseau partagé, *etc.*

Enfin, la description des ressources doit pouvoir être distribuée et obtenue via plusieurs sources distinctes. En effet, chaque administrateur d'un site de grille peut être responsable de générer la description des ressources de son site et de la diffuser par un moyen qu'il contrôle. Au vu de la quantité de ressources qui peuvent être disponibles dans une grille de calcul, la distribution de la description des ressources est un facteur qui permet le passage à l'échelle.

5.2.3 Description des paramètres de contrôle de l'utilisateur

Les paramètres de contrôle sont les exigences de l'utilisateur qui veut garder un certain niveau de contrôle sur le processus automatique de déploiement de son application. Les paramètres de contrôle ne sont pas nécessaires à l'exécution de l'application, sinon ils seraient inclus dans la description spécifique d'application. Ils ne sont pas non plus inhérents aux ressources de la grille. Ils expriment seulement les contraintes de l'utilisateur sur une exécution particulière d'une application.

5.2.3.1 Différents niveaux de contrôle

Les paramètres de contrôle peuvent exprimer des contraintes de plus ou moins haut niveau. Les paramètres de contrôle de « bas niveau » expriment des contraintes qui sont *directement* traduisibles en des choix par le planificateur de déploiement. Par exemple, les paramètres de contrôle peuvent imposer de ne lancer les processus de l'application que via l'intergiciel Globus ; dans ce cas, seules les ressources de calcul qui offrent cette méthode de soumission de tâches seront sélectionnées. D'autres exemples de paramètres de contrôle de bas niveau sont :

- définir un environnement particulier (variables d'environnement, répertoire courant, *etc.*) pour certains constituants de l'application ;
- interdire de lancer des processus à distance via SSH ;
- demander que les ressources choisies soient exclusivement dédiées à l'exécution de l'application de l'utilisateur ;
- imposer le degré de parallélisme d'une application MPI au lieu de laisser le planificateur prendre cette décision ;

– demander que tel programme de visualisation s'exécute suffisamment près de la station de travail de l'utilisateur (latence inférieure à 100μs entre la station de travail et la machine d'exécution du programme de visualisation) ;
– ne sélectionner que des ressources qui soient dans le domaine `.irisa.fr`, ou bien uniquement celles dont les processeurs sont les Sparc les plus rapides disponibles, ou encore celles qui ne sont pas payantes.

Les paramètres de contrôle de « haut niveau » ne sont pas directement traduisibles en choix par le planificateur. Par exemple, l'utilisateur peut demander que son application s'exécute dans un intervalle de temps minimal. Une telle requête est beaucoup plus difficile à satisfaire, car elle nécessite de connaître le *comportement* de l'application à l'exécution. Comme nous l'avons mentionné à la section 2.2.2.4 (page 15), ICENI tient compte du comportement et des performances des différentes implémentations des composants pour les sélectionner [86, 85]. Dans ce cas, il est nécessaire d'avoir un planificateur de déploiement particulièrement sophistiqué.

5.2.3.2 Relation avec le planificateur de déploiement

Les contraintes exprimées par les paramètres de contrôle peuvent être plus ou moins fortes : l'utilisateur doit pouvoir exiger une propriété absolument nécessaire, ou bien indiquer une préférence non indispensable. S'il est trop exigent, il se peut que le planificateur ne parvienne pas à trouver de solution et échoue à produire un plan de déploiement.

Enfin, les paramètres de contrôle peuvent sélectionner un planificateur de déploiement particulier. Comme le montre la section 5.3, il existe plusieurs algorithmes possibles pour implémenter un planificateur. L'outil de déploiement en sélectionne un par défaut si l'utilisateur n'en exige pas un particulier dans les paramètres de contrôle.

5.2.3.3 Re-déploiement

Même si le re-déploiement n'est pas l'objet de nos travaux (*cf.* section 4.1.2, page 54), nous remarquons que les paramètres de contrôle peuvent offrir une solution au re-déploiement d'une application qui aurait été interrompue, en raison d'une défaillance par exemple. Si l'application peut être migrée partiellement et relancée sur d'autres ressources, les paramètres de contrôle peuvent spécifier la liste des processus qui ont déjà été déployés (ainsi que leurs coordonnées), et auxquels devront se rattacher ceux qui restent à (re-)déployer. Ce mécanisme revient à effectuer un déploiement en plusieurs étapes.

5.2.4 Discussion

Cette section a présenté la liste des différentes informations qui sont nécessaires en entrée pour l'outil de déploiement automatique. Heureusement, l'utilisateur qui veut déployer son application n'a pas la charge de générer lui-même toutes ces informations : la description de l'application est constituée une fois pour toutes lors de la phase d'assemblage par son concepteur ; et la description des ressources de la grille est générée une fois pour toutes elle aussi par les administrateurs des différents sites de la grille. Le seul document que l'utilisateur peut avoir à produire est une description des paramètres de contrôle s'il a des exigences particulières sur le déploiement de son application.

Parmi ces informations nécessaires en entrée, nous avons plus particulièrement contribué à la description et au packaging des applications parallèles MPI (*cf.* section 7.1, page 119) et des applications mixtes GRIDCCM (*cf.* section 7.2, page 132), ainsi qu'à la description générique d'applications (*cf.* section 8.1, page 138) et la description de la topologie des réseaux complexes des grilles de calcul (*cf.* chapitre 6, page 95).

Enfin, nous remarquons que les paramètres de contrôle peuvent nécessiter des planificateurs de déploiement très sophistiqués. C'est le point que nous abordons dans la section suivante.

5.3 Planification du déploiement

Cette section présente en détail notre vision du plan de déploiement produit par le planificateur et des exemples simples d'algorithmes de planification.

5.3.1 Plan de déploiement

Suivant les paramètres de contrôle, soit la planification a une solution, soit elle n'en a pas. Si les contraintes imposées par l'utilisateur par le biais des paramètres de contrôle sont trop restrictives, alors le planificateur peut échouer à trouver des solutions et renvoyer une erreur. S'il trouve une solution, alors elle est consignée sous la forme d'un plan de déploiement. Ce plan de déploiement correspond à la liste des actions à effectuer pour lancer l'application. Il peut s'exprimer comme la projection des constituants de l'application sur les ressources sur lesquelles ils devront s'exécuter. Plus précisément, c'est un ensemble d'associations entre :

- une implémentation de chaque instance des programmes de l'application,
- et une méthode de soumission de tâche attachée à une ressource de la grille.

En effet, chaque programme d'une application peut avoir une cardinalité supérieure à un (*i.e.*, un constituant de l'application peut être instancié plusieurs fois), et le planificateur doit sélectionner, pour chaque instance de programme, l'implémentation qui sera utilisée sur la ressource sur laquelle elle sera lancée. De plus, le planificateur de déploiement doit aussi choisir le protocole de lancement de processus à distance pour les ressources qui en offrent plusieurs (SSH, Globus, UNICORE, *etc.*). En outre, le plan de déploiement peut indiquer dans quel *ordre* les instances de programmes doivent être lancées, si des dépendances l'exigent.

Dans notre architecture, l'*exécution du plan de déploiement* ne doit faire intervenir *aucun choix* : tout doit avoir été décidé par le planificateur. Une autre vision possible du plan de déploiement aurait pu être

- soit de ne pas spécifier la méthode de soumission de tâches pour les ressources, mais simplement d'associer les instances de programmes de l'application à des machines, en laissant le choix du protocole de lancement de processus au moment de l'exécution du plan de déploiement ;
- soit de spécifier une liste ordonnée de méthodes de soumission de tâches à essayer (dans l'ordre de préférence indiqué) lors de l'exécution du plan.

Dans le cas de la défaillance d'une méthode de soumission de tâches pour une ressource (le daemon SSH n'a pas été lancé, par exemple), l'avantage de cette approche est que l'exécuteur du plan de déploiement peut recourir au protocole de lancement de processus suivant dans la liste de ceux supportés par la ressource, puisque ce protocole n'est pas imposé par le plan

de déploiement. Ainsi, l'outil de déploiement ne reporte pas d'échec parce qu'il n'a pas choisi la « bonne » méthode de soumission. En revanche, cette approche présente l'inconvénient de rendre l'exécution du plan de déploiement plus compliquée et moins systématique. En effet, l'utilisateur peut imposer ou interdire certaines méthodes de soumission par le biais des paramètres de contrôle : par exemple, il peut interdire l'utilisation d'UNICORE pour lancer des processus à distance si son certificat d'utilisateur UNICORE n'est plus valide. Dans ce cas, le plan de déploiement devra, à la différence du cas général, imposer une ou plusieurs méthodes de soumission de tâches pour chaque ressource qui offrirait l'interface d'UNICORE pour lancer des processus.

Dans un souci de simplicité, nous avons décidé que le plan de déploiement indique *systématiquement* une méthode précise de soumission de tâches pour chaque ressource sélectionnée. En cas de défaillance d'un protocole de lancement de processus pour une ressource donnée, l'exécuteur du plan de déploiement peut rapporter un message d'erreur indiquant le protocole et la ressource incriminés. Ensuite,

- soit l'outil de déploiement reboucle sur la planification de déploiement en ajoutant un paramètre de contrôle qui interdise l'emploi de la méthode de soumission défaillante sur la ressource concernée,
- soit l'utilisateur est avisé de l'échec et il peut relancer le déploiement de son application en ajoutant le même paramètre de contrôle.

L'inconvénient de notre approche est de nécessiter une deuxième planification du déploiement en cas de défaillance d'un protocole de lancement sur une ressource, alors que cette replanification est évitée dans l'autre approche, si jamais la ressource offre plusieurs méthodes de soumission.

Dans tous les cas, l'outil de déploiement doit être capable de *défaire* (« *roll-back* ») les actions du plan de déploiement déjà effectuées. Si l'unique protocole de lancement de processus sur une ressource échoue (soit parce que la ressource n'en offre pas d'autre, soit parce qu'il est imposé par le plan de déploiement), alors il faut annuler les processus déjà lancés, et désinstaller tous les fichiers sur les ressources. Comme annoncé à la section 5.1.3, nous ne gérons pas, dans notre architecture, la terminaison et la désinstallation des applications ; en revanche, toutes les informations sont disponibles pour ce faire dans le rapport de déploiement.

5.3.2 Algorithmes de planification

5.3.2.1 Rôles du planificateur

En plus des responsabilités énumérées à la section 5.1.2.1, le planificateur peut aussi avoir pour rôle de déterminer le nombre d'instances d'un programme à lancer. C'est le cas notamment pour des applications parallèles qui peuvent s'exécuter avec un nombre variable de processus : il appartient alors au planificateur de décider du degré de parallélisme s'il n'est pas imposé par l'utilisateur (par le biais des paramètres de contrôle). Les différents rôles du planificateur ne sont pas forcément indépendants : les fonctions de

1. sélection des ressources,
2. sélection d'une méthode de soumission de tâches,
3. placement des instances de programmes,
4. détermination de la cardinalité de ces instances si nécessaire,

5. et sélection des implémentation des programmes,

ne sont pas exécutées séquentiellement dans cet ordre. Par exemple, la sélection des ressources dépend des implémentations disponibles, puisque le planificateur doit veiller à la compatibilité entre les ressources de calcul et les versions compilées des programmes en termes de système d'exploitation et d'architecture matérielle.

5.3.2.2 Implémentations

Plusieurs implémentations de planificateurs peuvent se « brancher » dans notre architecture de déploiement, mettant en œuvre différents algorithmes sous la forme de plugins. Cette modularité permet par exemple de tester et d'évaluer différents algorithmes de planification, du point de vue du temps qu'il mettent à produire un plan de déploiement, ou du point de vue de l'efficacité de leurs choix de placement (en comparant les temps d'exécution de l'application, par exemple).

Il se peut que le planificateur soit déterministe ou non, qu'il trouve une solution optimale ou simplement satisfaisante, qu'il soit plus ou moins sophistiqué pour tenir compte de paramètres de contrôle plus ou moins élaborés, *etc.* Deux exemples simples d'algorithmes de planification que nous avons mis en œuvre sont :

***Random* :** cet algorithme de planification, schématisé sur la figure 5.2, place les instances de programmes de l'application *au hasard* sur les ressources, en respectant tout de même les contraintes de compatibilité de système d'exploitation et d'architecture matérielle entre les machines et les implémentations sélectionnées ;

***Round-robin* :** cet algorithme de planification, schématisé sur la figure 5.3, place les instances de programmes de l'application sur les ressources *au fur et à mesure* qu'elles sont découvertes, et en respectant les contraintes de compatibilité de système d'exploitation et d'architecture matérielle.

Un avantage de l'algorithme *round-robin* est qu'il n'a, en général, pas besoin de découvrir l'intégralité des ressources disponibles sur la grille, donc il passe bien à l'échelle. En revanche, ces deux algorithmes ne sont pas assez sophistiqués pour satisfaire des contraintes de communication entre les programmes, en regroupant sur un même site de calcul les processus qui communiquent intensivement par exemple.

PSF / Sekitei. Le système Sekitei du projet PSF (*Partitionable Services Framework*, [98]) est un planificateur de déploiement de services à base de composants. Sekitei est fondé sur des techniques d'intelligence artificielle [105, 106] : il tient compte des propriétés des composants à déployer, et de l'état courant de l'environnement de déploiement (ressources de calcul et réseau) ; il sait aussi sélectionner les implémentations des composants. Les résultats de Sekitei sont tout à fait complémentaires des nôtres et pourraient s'insérer dans notre architecture de déploiement.

Pegasus. Pegasus (*cf.* section 4.3.3.5, page 67, [59]) possède un planificateur de déploiement nommé Prodigy [159], et issu de la recherche en intelligence artificielle. Le planificateur de Pegasus sait projeter des workflows de tâches séquentielles et individuelles, en tenant compte de la localisation des données manipulées par les tâches. Pegasus est flexible dans le sens où il permet de brancher plusieurs algorithmes de planification sous la forme de plugins.

```
 1. Pour chaque programme P de l'application
 2.    Pour chaque instance à déployer pour le programme P
 3.       Vider la liste L de machines déjà envisagées
 4.       Choisir une machine M au hasard parmi celles disponibles hors
               de la liste L
 5.       S'il reste une telle machine M hors de la liste L,
 6.          Alors ajouter la machine M à la liste L
 7.          Sinon il n'y a pas de solution: échec de la planification
 8.       Pour chaque implémentation I du programme P
 9.          Si le système d'exploitation et l'architecture matérielle de
                la machine M sont compatibles avec l'implémentation I,
10.             Alors ajouter au plan de déploiement l'association entre
                   l'implémentation I et une méthode de soumission
                   quelconque de la machine M, et continuer en 2.
11.             Sinon continuer en 4.
12. Planification terminée avec succès
```

FIG. 5.2 – Algorithme simplifié de planification « *random* ».

```
 1. Tant qu'il reste des instances de programme P à placer
 2.    Découvrir la machine suivante disponible M (ou bien revenir à la
            première machine découverte)
 3.    Pour chaque implémentation I du programme P
 4.       Si le système d'exploitation et l'architecture matérielle de
                la machine M sont compatibles avec l'implémentation I,
 5.          Alors ajouter au plan de déploiement l'association entre
                   l'implémentation I et une méthode de soumission de
                   la machine M, et continuer en 1.
 6.          Sinon continuer en 3.
 7.    Si aucune implémentation I ne peut être placée sur la machine M
            et que toutes les machines disponibles n'ont pas encore été
            envisagées pour le programme P
 8.       Alors continuer en 2.
 9.       Sinon il n'y a pas de solution: échec de la planification
10. Planification terminée avec succès
```

FIG. 5.3 – Algorithme simplifié de planification « *round-robin* ».

Même si le planificateur de Pegasus se contente de placer des tâches simples d'applications non concurrentes, il démontre que l'idée de recourir à des techniques d'intelligence artificielle est bonne.

Isolation du planificateur. La planification de déploiement est une opération complexe, qui doit être conçue et implémentée par un spécialiste. C'est pourquoi nous avons choisi d'isoler cette phase dans notre architecture de déploiement automatique (*cf.* figure 5.1). Pour ce faire,

- nous avons introduit la notion de plan de déploiement pour s'interfacer avec l'exécuteur du plan,
- et la section 8.2.2 (page 155) liste les opérations élémentaires que nous avons identifiées pour interfacer le planificateur avec la description (générique) de l'application, des ressources et des paramètres de contrôle.

5.3.3 Discussion

Grâce à la notion de description générique d'application, qui permet de décrire tout type d'application dans un unique formalisme, la planification de déploiement est indépendante du type de l'application à déployer : un seul planificateur peut servir pour déployer des applications distribuées CCM, des applications parallèles, ou bien des applications mixtes.

Notre architecture suppose que les ressources ne changent pas d'état entre le moment où elles sont sélectionnées par le planificateur et celui où le plan de déploiement est exécuté : les charges des processeurs sont peu modifiées, les ressources restent disponibles, *etc.* Pour faire face à cette limitation, nous imaginons deux solutions qui pourraient être envisagées :

Réservation transactionnelle : un mécanisme de réservation des ressources choisies pourrait être initié par le planificateur de déploiement, afin de s'assurer que les machines soient toujours disponibles au moment de l'exécution du plan de déploiement (et donc du lancement des processus).

Re-déploiement : si l'exécuteur du plan de déploiement constate que le statut ou la disponibilité des machines ont changé au moment du lancement des processus, alors il peut le signaler et relancer le processus de planification (*cf.* section 5.2.3.3) ; comme le système d'information sur les ressources est dynamique, le planificateur trouvera une solution différente si l'état des ressources a été modifié.

Comme les algorithmes de planification sortent du cadre de nos compétences, nous n'avons pas contribué à la conception de planificateurs plus élaborés que *random* et *round-robin*. La planification hérite des travaux sur l'ordonnancement et sur le placement de tâches, à condition de les enrichir des fonctions de sélection des ressources et de sélection des implémentations.

Grâce à la flexibilité de notre architecture de déploiement, le plan de déploiement produit par le planificateur peut être intercepté, analysé et modifié par l'utilisateur, ou bien il peut être directement envoyé à l'exécuteur du plan de déploiement, comme le détaille la section suivante.

5.4 Exécution du plan de déploiement

Cette section énumère les différentes étapes qui constituent l'« exécution du plan de déploiement » : installation des fichiers sur les ressources d'exécution sélectionnées, lancement des processus, et configuration de l'application. L'exécuteur prend en entrée un plan de déploiement tel qu'il a été défini à la section 5.3.1 : la phase d'exécution du plan ne fait intervenir aucun choix, puisque le plan spécifie l'intégralité des actions à effectuer, ainsi que l'ordre dans lequel elles doivent l'être.

Cette section présente également le contenu du rapport de déploiement produit par l'exécuteur du plan. Le rôle du rapport de déploiement est double : d'une part, il permet de savoir si l'exécution du plan s'est bien passée ou si elle a échoué (et pour quelle raison, sur quelle action) ; d'autre part, il contient toutes les informations pour désinstaller les fichiers et stopper l'exécution de l'application. Le rapport de déploiement peut se contenter d'enrichir le plan de déploiement en l'annotant avec l'état d'avancement des opérations à effectuer.

5.4.1 Installation des fichiers

L'installation des fichiers comprend le transfert des fichiers, ainsi que leur mise en place dans un répertoire approprié et sous un nom indiqué par le plan de déploiement : ce dernier spécifie la source et la destination des fichiers. Les fichiers concernés sont les binaires des programmes de l'application, les fichiers de données en entrée, les dépendances (bibliothèques, *etc.*).

Le protocole de transfert à utiliser est précisé par le plan de déploiement : il repose directement sur des intergiciels d'accès aux ressources de grilles (GridFTP[3] de Globus, par exemple), ou bien sur des bibliothèques de plus haut niveau, telles que Elagi (voir la section 4.3.1.2, page 64). La réussite ou l'échec de l'installation de chaque fichier est consigné dans le rapport de déploiement. En cas d'échec, les raisons de l'échec sont explicitées : fichier source absent, protocole inaccessible, impossible de placer le fichier à destination (manque d'espace disque, permissions insuffisantes), *etc.*

5.4.2 Lancement des processus de l'application

Le lancement des processus à distance, via le protocole indiqué par le plan de déploiement, consiste plus exactement à *soumettre une tâche* auprès d'un intergiciel ou d'un système de batch. Il n'y a aucune garantie sur l'*intervalle de temps qui peut s'écouler entre la soumission et le lancement effectif des processus*. Par exemple, il se peut que certains processus de l'application soient lancés immédiatement sur un site de la grille, tandis que d'autres processus ne seront lancés qu'au bout de plusieurs heures sur un autre site. Dans ce cas, il se peut que les premiers processus lancés aient été terminés par le système de batch qui gère les ressources, car le temps imparti pour leur exécution est arrivé à expiration. Cependant, il s'agit là d'un problème afférent aux intergiciels d'accès et à la qualité de service qu'il sont capables d'offrir (réservation, co-allocation de ressources).

Le plan de déploiement peut spécifier qu'un environnement soit configuré avant de lancer les processus, en définissant des variables d'environnement, en positionnant le répertoire

[3] Voir la section 3.3.2.1, page 46.

courant, *etc.* (configuration « *avant lancement* »). Ensuite, le processus est invoqué avec les arguments en ligne de commande précisés par le plan de déploiement.

Pour chaque tâche soumise, le rapport de déploiement indique si le lancement a eu lieu avec succès ou bien s'il a échoué ; il précise également quelles sont les coordonnées des constituants de l'application effectivement lancés, telles que les PID des processus, les numéros de tâches soumises via Globus, les références d'objets (IOR) pour les applications CCM, *etc.* Ces informations permettent d'interroger l'état des tâches (en cours d'exécution, terminée, suspendue, *etc.*), et de contrôler les tâches (suspendre, redémarrer, annuler, *etc.*).

5.4.3 Configuration de l'application *après lancement*

À ce stade, le rapport de déploiement est annoté avec les coordonnées des constituants de l'application pour pouvoir les contacter et les configurer. Les phases précédentes étaient indépendantes du type d'application en cours de déploiement, tandis que la phase de configuration après lancement est spécifique à la nature de l'application.

À titre d'exemple, pour une application CCM, la phase de configuration après lancement consiste à charger les DLL dans les serveurs de conteneurs (*ComponentServers*), à interconnecter les composants entre eux comme indiqué dans la description *spécifique* d'application, à fixer les valeurs initiales de leurs paramètres, à appeler la méthode `configuration_complete()` pour chaque composant, et à activer l'application par appel de la méthode `run` de l'ORB. Cette suite d'actions est bien spécifique au type de l'application en cours de déploiement, et il est nécessaire de revenir à la description spécifique de l'application.

5.4.4 Discussion

Les étapes de la phase d'exécution du plan de déploiement ne sont pas nécessairement réalisées de manière indépendante les unes des autres. Par exemple, un intergiciel tel que Globus permet d'installer des fichiers et de soumettre une tâche sur une ressource de calcul donnée en une seule opération. C'est le rôle de l'exécuteur du plan de déploiement de s'interfacer le plus efficacement possible avec les divers intergiciels d'accès aux ressources des grilles de calcul.

En cas d'échec lors de l'exécution du plan de déploiement, l'exécuteur cesse immédiatement et consigne son rapport d'erreur dans le rapport de déploiement. Si l'outil de déploiement est incapable de contourner l'erreur, le rapport d'échec est retourné à l'utilisateur : c'est le cas par exemple lorsqu'un fichier source est introuvable, ou lorsqu'un fichier binaire se révèle ne pas être exécutable. En revanche, dans le cas d'un problème de protocole de transfert de fichiers inopérant par exemple, l'outil de déploiement peut relancer la planification en interdisant ce protocole de transfert précis sur la ressource incriminée. Comme mentionné à la section 5.3.1, notre architecture de déploiement ne gère pas la terminaison et la désinstallation des applications (*roll-back*), même si toutes les informations sont disponibles dans le rapport de déploiement pour contacter les ressources sur lesquelles des fichiers ont été installés et des processus ont été lancés.

Enfin, la description générique d'application permet de n'implémenter
- qu'un seul planificateur de déploiement,
- et qu'un seul exécuteur de plan de déploiement,

pour déployer tout type d'application. En revanche, la phase de configuration après lancement de l'application nécessite de revenir à la description *spécifique* de l'application (*cf.* section 9.4, page 172).

5.5 Conclusion

Cette section a présenté notre solution pour simplifier le déploiement d'applications de types variés, voire mixtes, sur les infrastructures complexes que sont les grilles de calcul. Nous avons décrit l'architecture générale, modulaire et flexible, de l'outil intégré de déploiement automatique d'applications, en montrant que le planificateur de déploiement se trouve au cœur de ce modèle. Nous avons également listé les informations nécessaires en entrée de l'outil de déploiement : l'utilisateur a très peu d'informations à générer lui-même, à savoir les paramètres de contrôle. Enfin, nous avons énuméré les étapes de l'exécution du plan de déploiement pour lancer effectivement l'application.

Nous avons souligné quelques limites de notre modèle. Notre architecture ne gère pas l'intégralité du cycle de vie de l'application, en particulier en ce qui concerne la terminaison et la désinstallation de l'application. Cependant, toutes les informations nécessaires à ces opérations sont disponibles dans le rapport de déploiement. Une autre limitation que nous n'avons pas résolue, mais pour laquelle nous avons envisagé des pistes, est la modification de l'état des ressources entre le moment de la planification et celui de l'exécution du plan de déploiement.

Le désir de simplement taper « `grid-run` » pour lancer une application n'est pas nouveau [145] : il est conforme à l'objectif de transparence d'utilisation à long terme des grilles de calcul (*cf.* section 3.1.4.1, page 41). Ici, nous avons proposé une solution à cet objectif. Nous remarquons que ce modèle d'architecture n'est pas spécifique aux grilles de calcul : il peut également s'appliquer à des infrastructures d'exécution plus simples. Dans tous les cas, il permet de simplifier la tâche de l'utilisateur, et d'*accélérer* le processus de déploiement, par rapport à un processus manuel.

Les quatre chapitres suivants présentent en détail les points particuliers de cette architecture sur lesquels nous avons le plus significativement contribué : la description de la topologie des réseaux complexes rencontrés dans les grilles de calcul, la description d'applications (description spécifique d'applications MPI et GRIDCCM, puis description générique d'applications de tous types), et la configuration des applications pour qu'elles s'adaptent à leurs environnements d'exécution.

Chapitre 6

Description des ressources réseau

Sommaire

Le chapitre précédent a montré que l'outil de déploiement automatique a besoin d'une description des ressources disponibles de la grille de calcul sur laquelle l'utilisateur veut déployer son application. Plus les ressources sont décrites avec précision et meilleurs seront les choix du planificateur de déploiement en matière de placement de l'application. Ce chapitre commence par montrer l'intérêt de pouvoir décrire les réseaux de communication entre les ressources, puis il présente notre modèle de description de topologie réseau.

6.1 Identification des informations utiles sur les ressources

Cette section identifie les deux types de ressources que nous avons besoin de décrire : d'une part les nœuds de calcul et de stockage, et d'autre part les liens réseau entre ces nœuds, c'est-à-dire leurs propriétés de performance et leur topologie.

6.1.1 Ressources de calcul et de stockage

Le planificateur de déploiement est responsable de sélectionner les implémentations des programmes qui sont compatibles avec les systèmes d'exploitation et les architectures matérielles des ressources. Ces deux informations sont donc évidemment nécessaires dans la description des ressources. Le planificateur de déploiement est également responsable de déterminer, si besoin, le nombre d'instances des programmes de l'application à déployer. Pour ce faire, il peut avoir besoin de connaître le nombre de processeurs des machines disponibles, le nombre de nœuds au sein des clusters, *etc.*

Les exigences de l'utilisateur, par l'intermédiaire des paramètres de contrôle, peuvent nécessiter d'avoir accès à la fréquence des processeurs, la taille mémoire physique disponible, *etc.* ainsi qu'à des informations sur les ressources de stockage attachées aux nœuds de calcul. Par exemple, ces informations portent sur l'espace disque disponible, l'existence ou non de quotas (et leur valeur), la localisation et le type (NFS[1], PVFS[2], *etc.*) des points de montage, *etc.*

En plus des informations *matérielles* sur les nœuds de calcul, il peut être utile de connaître leurs caractéristiques *logicielles*, afin de vérifier que les dépendances des programmes seront satisfaites. Par exemple, un programme peut nécessiter qu'une JVM soit installée, ou bien qu'une bibliothèque particulière soit disponible dans une version précise.

Diversité des méthodes de soumission de tâches. La description des ressources doit spécifier la ou les méthodes de soumission de tâches sur les machines de la grille (intergiciels d'accès aux ressources de grille tels que Globus ou UNICORE, systèmes de batch tels que PBS ou SGE, protocole SSH, *etc.*), ainsi que les protocoles de transfert de fichiers (GridFTP, SCP, *etc.*).

Certaines ressources peuvent présenter plusieurs méthodes d'accès : par exemple, un cluster peut être accessible par Globus ou par PBS. Mais les ressources peuvent également être considérées avec des granularités différentes. La figure 6.1 illustre un cluster dont chaque machine peut être accédée *individuellement* par le protocole SSH, ou *globalement* via le système de batch PBS. Chaque point d'accès SSH permet de soumettre une tâche sur une unique machine, alors que PBS donne accès à plusieurs machines. La description des ressources doit bien spécifier que les machines peuvent être accédées via SSH ou PBS à des granularités différentes, mais qu'elles représentent le *même* ensemble de machines.

En plus des informations sur les nœuds de calcul et de stockage, la section suivante énumère les configurations réseau qui doivent pouvoir être décrites pour que le planificateur de déploiement puisse placer des programmes qui communiquent entre eux.

[1] NFS : *Network File System*, un système de fichier partagé via un réseau local.

[2] PVFS : *Parallel Virtual File System*, un système de fichier partagé pour les clusters, et optimisé pour les entrées-sorties en parallèle.

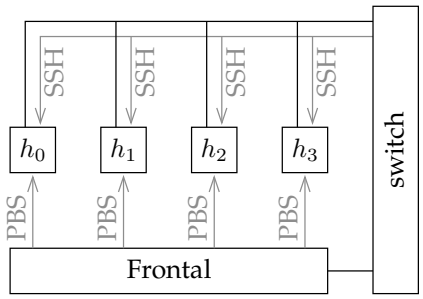


FIG. 6.1 – Grille E : cluster sur les nœuds duquel une tâche peut être soumise *individuellement* à chaque nœud par le protocole SSH, ou bien *globalement* via le système de batch PBS.

6.1.2 Connexions réseau entre les nœuds

6.1.2.1 Les besoins du planificateur de déploiement

Pour pouvoir satisfaire toutes les contraintes liées à une application (tel processus doit pouvoir communiquer directement avec tel autre processus) et celles liées aux exigences de l'utilisateur par le biais des paramètres de contrôle, le planificateur de déploiement a besoin d'informations précises sur les ressources réseau. Par exemple, l'utilisateur peut demander que son application MPI s'exécute :

- sur 32 machines interconnectées par un réseau Myrinet-2000 ;
- ou bien sur 64 machines reliées par un réseau de débit au moins égal à 1 Gb/s, et que la latence entre ces 64 machines et un nœud de visualisation d'adresse IP 129.88.70.61 soit au plus de 100 ms.

Pour satisfaire ce genre de requête, le planificateur de déploiement doit avoir accès à une description précise des performances et de la topologie des réseaux de communication entre les machines de la grille.

6.1.2.2 Performances et topologie

Essentiellement, nous distinguons deux catégories d'informations sur les ressources réseau d'une grille de calcul : les propriétés de performance et la topologie.

Les propriétés de performance des réseaux concernent les valeurs numériques des débits, latences, gigues, taux de perte des liens, *etc.*, que ce soient leurs valeurs moyennes, instantanées, ou encore leurs variances dans le temps. Le groupe de travail NMWG (*Network Performance Measurement Working Group*, [117]) du GGF fournit des solutions pour décrire ces informations sur les propriétés de performances des réseaux, en les enrichissant de la description des méthodologies de mesure employées pour obtenir les résultats [119]. De plus, le NMWG propose une analyse intéressante de la classification des caractéristiques réseau.

Cependant, le NMWG ne décrit pas la topologie réseau de manière synthétique. La description suivant le NMWG spécifie quelles machines sont connectées à quelles autres machines pour communiquer, quels sont les protocoles permis, les ports ouverts, les technologies réseau, *etc.* Les sections suivantes donnent quelques exemples de topologies réseau que nous

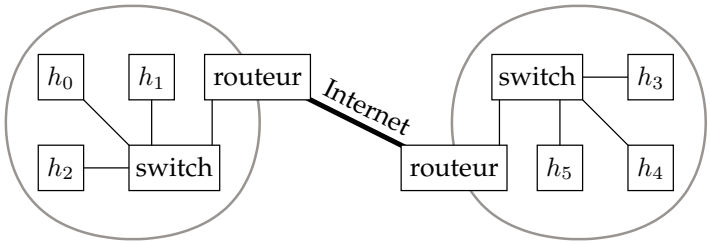


FIG. 6.2 – Grille A : exemple de grille de calcul simple.

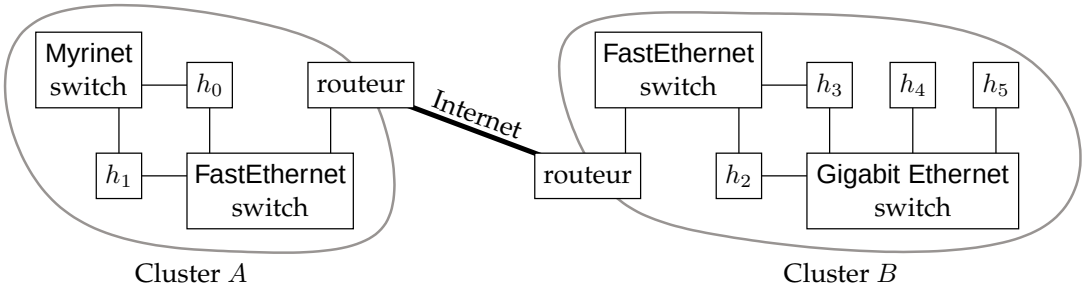


FIG. 6.3 – Grille B : recouvrement de différentes technologies réseau.

avons besoin de savoir décrire pour permettre au planificateur de déploiement de placer les applications concurrentes.

6.1.2.3 Réseaux simples

La figure 6.2 illustre une grille de calcul très simple, constituée de deux clusters reliés par Internet. À l'intérieur de chaque cluster, trois nœuds h_i sont reliés entre eux par un réseau IP FastEthernet. Les routeurs ne pratiquent aucun filtrage, et les nœuds de calcul peuvent communiquer directement les uns avec les autres. Un réseau aussi simple doit pouvoir être décrit *simplement*.

6.1.2.4 Appartenance à plusieurs réseaux

La figure 6.3 montre une grille de calcul constituée de deux clusters interconnectés par Internet. Chacun de ces clusters possède deux réseaux de technologies différentes. Le cluster A comprend un réseau Myrinet et un réseau FastEthernet ; le cluster B contient un réseau Gigabit Ethernet et un réseau FastEthernet. Dans le cluster B, les cartes réseau Gigabit Ethernet possèdent des adresses IP différentes de celles du réseau FastEthernet, et qui ne peuvent pas être routées depuis l'extérieur de ce réseau. Dans le cluster A, Myrinet n'est pas un réseau IP : le pilote qui donne accès à ce matériel peut être, par exemple, GM [126], BIP [138] ou MX [127].

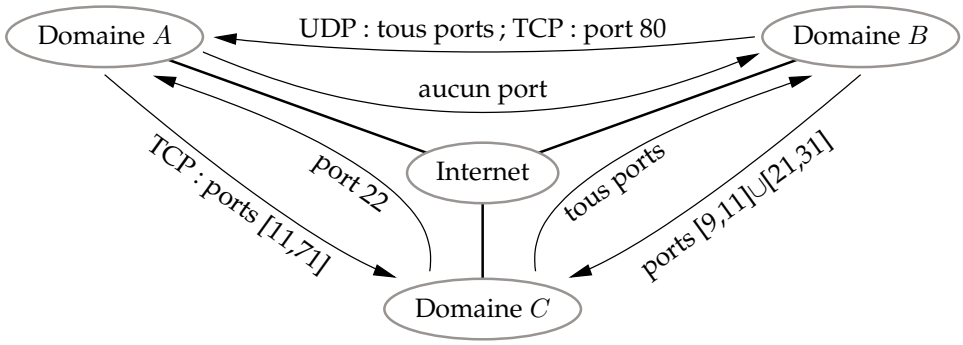


FIG. 6.4 – Grille C : filtrage du trafic réseau par des pare-feux ; les ports ouverts sont spécifiés.

6.1.2.5 Pare-feux, NAT et liens asymétriques

Les grilles de calcul s'étendent sur plusieurs domaines d'administration. Pour des raisons de sécurité, les réseaux locaux sont très souvent protégés contre les intrusions par des pare-feux (« *firewalls* » en anglais), qui filtrent les communications. Un pare-feux peut fermer, de manière sélective, certains ports TCP ou UDP pour le trafic réseau en provenance ou à destination de certains domaines. La figure 6.4 illustre les règles de filtrage qu'un pare-feu peut imposer sur les communications entre les différents sites d'une grille : ces règles dépendent du sens des communications et des domaines distants.

La traduction d'adresses (NAT, *Network Address Translation*, [149]) est un autre mécanisme de sécurité ; il est également utilisé lorsque la plage d'adresses IP publiques allouées à un site est trop petite (le site ne possède pas suffisamment d'adresses IP). Dans ce cas, des adresses IP *privées* sont affectées à certaines machines, et une ou plusieurs passerelles sont responsables de relayer les communications en provenance et à destination des machines sous adresses IP privées[3].

Le cas des adresses IP *publiques non routables* est similaire à celui des adresses IP privées. Par exemple, sur la figure 6.3, les nœuds h_4 et h_5 peuvent être reliés au switch Gigabit Ethernet avec des cartes réseau sous adresses IP publiques sans que ces adresses IP soient routables depuis Internet. Ce cas de figure doit pouvoir être décrit en spécifiant que les nœuds h_2 et h_3 servent de relais.

L'information sur les pare-feux et sur les passerelles NAT peut être utile à l'outil de déploiement, tant pour le placement des applications que pour le lancement éventuel d'un mécanisme de relais pour ré-émettre les messages vers des machines non accessibles depuis Internet par exemple.

Les pare-feux sur chaque site rendent les communications asymétriques : le trafic réseau autorisé à destination d'un site n'est pas forcément le même que celui qui est permis en provenance de ce site. Cette asymétrique peut également concerner les performances réseau, telles que le débit. C'est le cas par exemple des connexions ADSL (*Asymmetric Digital Subscriber*

[3]Pour les connexions sortantes, *i.e.* en provenance du réseau local sous adresses IP privées, une passerelle NAT modifie l'entête IP en remplaçant l'adresse IP privée de la machine source par l'adresse IP publique de la passerelle, et elle conserve en mémoire la trace de cette connexion. Ainsi, lors de la réponse en provenance d'Internet, la passerelle saura rediriger la communication vers la machine destination sous adresse IP privée en modifiant l'entête IP du message.

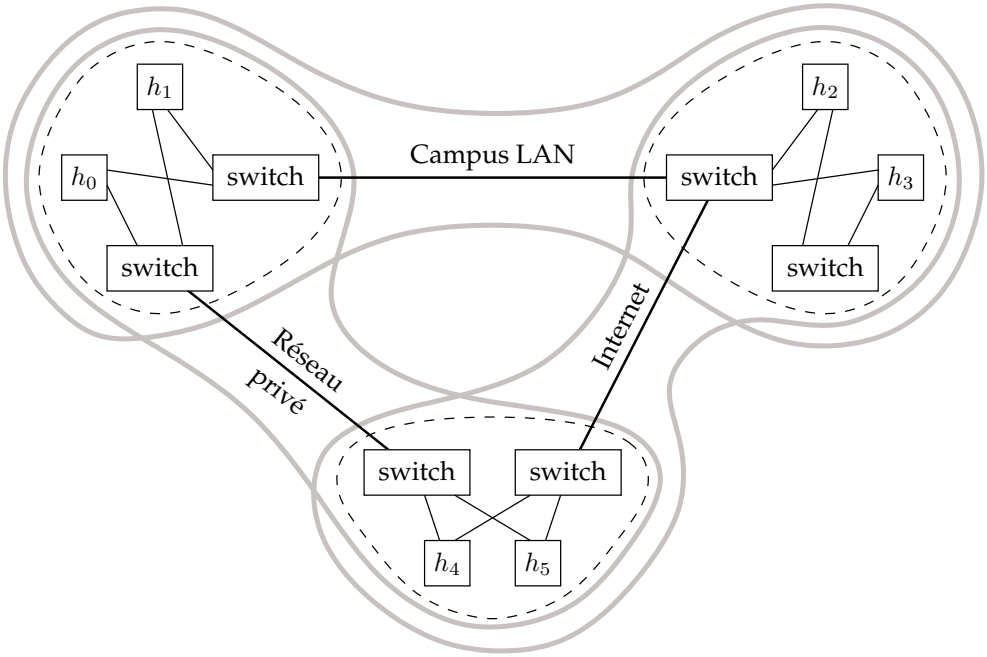


FIG. 6.5 – Grille D : exemple de réseau non-hiérarchique.

Line), qui n'offrent pas le même débit dans le sens montant que dans le sens descendant. La description de la topologie réseau doit permettre de décrire ce type d'asymétrie.

Ainsi, une description réaliste des ressources réseau doit comprendre les pare-feux, les réseaux en adresses privées avec leurs passerelles de traduction d'adresses, ainsi que la description de liens asymétriques.

6.1.2.6 Topologies réseau non-hiérarchiques

La figure 6.5 illustre un exemple de *réseau non-hiérarchique*. Cette configuration est constituée de trois clusters FastEthernet. Deux clusters sur les trois sont connectés à Internet, une autre paire de clusters est interconnectée par un réseau privé haute performance (tel que le réseau Grid'5000 en France, *cf.* section 3.2.2, page 43), et deux autres clusters sont reliés à un réseau local de campus.

Ce type de configuration est complexe mais réaliste : il doit pouvoir être décrit sans énumérer une par une les connexions réseau. En plus d'être fastidieuse et difficile à maintenir, cet énumération des liens individuels ne passe pas à l'échelle.

En outre, le planificateur de déploiement n'a pas besoin de connaître la topologie *physique* du réseau, c'est-à-dire la description de chaque lien physique, chaque câble réseau, chaque routeur, chaque switch, *etc.* Le planificateur a seulement besoin de connaître la topologie *logique* du réseau, qui indique quelles machines sont interconnectées, et quelles sont les caractéristiques de ces interconnexions. Cette information de plus haut niveau et plus synthétique est plus facilement et directement exploitable par le planificateur de déploiement.

6.1.3 Travaux apparentés

Cette section présente l'état des travaux de recherche concernant la description des ressources. Ces travaux sont évalués par rapport aux besoins exprimés dans les sections précédentes, concernant la description des nœuds de calcul et de stockage, ainsi que des réseaux et leurs topologies.

6.1.3.1 Le MDS de Globus

Le MDS de Globus (*cf.* section 3.3.2.1, page 46) décrit les informations sur les nœuds de calcul et de stockage, ainsi que les interfaces de connexion réseau attachées à chaque nœud : adresses IP, types d'interface (Ethernet, FastEthernet, Gigabit Ethernet, *etc.*). De plus, les informations du MDS peuvent être mises à jour dynamiquement au fur et à mesure de leur évolution. Cependant, ces informations ne suffisent pas à décrire la topologie réseau, car elles ne spécifient pas *à quelles autres machines les interfaces sont reliées*.

6.1.3.2 GridLabMDS

GridLabMDS [19] fait partie du projet GridLab (*cf.* section 4.3.1.1, page 64). GridLabMDS repose sur le MDS de Globus, tout en reconnaissant que le MDS manque d'information sur les ressources réseau et sur les logiciels installés sur les machines [18]. Ainsi, GridLabMDS étend le MDS de Globus pour décrire les logiciels installés et les pare-feux en spécifiant les ports ouverts et fermés sur chaque machine. Cependant, cette description est insuffisante, car en pratique les ports sont ouverts ou fermés *dans un sens bien précis* (connexions entrantes ou sortantes) et, parfois, *vis-à-vis de sites précis* qu'il faut spécifier. De plus, GridLabMDS ne décrit pas la topologie réseau, ni la nature des liens réseau.

6.1.3.3 Le RSD de ZIB

RSD (*Resource Service Description*, [42, 103]) est une architecture logicielle pour décrire, enregistrer, et accéder aux ressources et services d'environnements de calcul complexes et hétérogènes. RSD est développé au *Konrad-Zuse-Zentrum für Informationstechnik* à Berlin (ZIB).

RSD décrit les caractéristiques de performances et les interconnexions des ressources réseau, sans se restreindre aux réseaux IP. RSD peut décrire n'importe quelle topologie réseau en spécifiant chaque lien physique entre chaque paire d'ordinateurs, chaque routeur, chaque switch, *etc.* Cependant, RSD ne permet de décrire que des réseaux purement hiérarchiques. De plus, RSD ne tient pas compte des pare-feux, et la description de la topologie physique est de trop bas niveau pour être commodément exploitable par le planificateur, et pour passer à l'échelle dans un environnement de grilles de calcul.

6.1.3.4 Le NMWG du GGF

Le NMWG du GGF (*cf.* section 6.1.2.2) permet de bien décrire les *caractéristiques de performance* des réseaux de grilles, telles que les performances numériques (débit, latence, gigue, taux de pertes). Cependant, il ne présente pas de solution satisfaisante pour la description

de la *topologie réseau*. NMWG décrit les liens physiques un par un, et donne beaucoup trop de détails sur la connectivité *physique*, les routeurs, les switches, *etc.* Cette information est de trop bas niveau pour être facilement exploitable par le planificateur de déploiement, qui a besoin d'une description de la topologie *logique* (et non physique) des réseaux. En revanche, nos travaux sur la description de la topologie réseau (*cf.* section 6.2) sont complémentaires des résultats de NMWG sur la description des caractéristiques de leurs performances (description des valeurs numériques de performances).

6.1.3.5 Remos

Remos (*REsource MOnitoring System*, [118]) est un logiciel qui permet aux applications de tenir compte de la topologie réseau sous-jacente sur laquelle elles s'exécutent, en obtenant les informations sur le réseau dont elles ont besoin. Remos représente la topologie réseau logique par un graphe. Les nœuds du graphe sont les ordinateurs, et les arêtes sont les liens réseau. Remos obtient la topologie de réseaux IP uniquement sur des réseaux locaux (LAN), en réalisant des mesures de performances. Remos n'est pas adapté aux grilles de calcul, car il ne considère pas les réseaux longue distance, les réseaux non IP, les pare-feux, *etc.*

6.1.3.6 ENV et GridML de SDSC

ENV (*Effective Network Views*, [147]) est un projet mené à SDSC (*San Diego Supercomputer Center*). ENV utilise un langage au format XML appelé GridML [186] pour décrire les réseaux. ENV est restreint aux clusters d'ordinateurs interconnectés par des réseaux locaux (LAN) sous IP. Ainsi, ENV n'est pas adapté à décrire les réseaux complexes et variés des grilles de calcul.

6.1.3.7 GridG et la base de données relationnelle RGIS

GridG [120] est un générateur de topologies réseau qui permet de simuler de façon réaliste les ressources de grilles, afin d'évaluer des intergiciels. GridG ne considère que des réseaux IP purement hiérarchiques, et il ne tient pas compte des pare-feux.

RGIS [66] est un service d'information sur les ressources sous la forme d'une base de données relationnelle. Cette approche permet d'obtenir une information relativement synthétique sur les ressources de grille, par le biais de requêtes SQL, en utilisant des jointures pour réaliser des sélections fines sur les propriétés des ressources. Cependant, RGIS décrit la topologie réseau en spécifiant chaque lien réseau individuel entre chaque paire de machines. De plus, les jointures sur de nombreuses tables de base de données distribuées présentent des problèmes de performance. Comme RGIS repose sur GridG, il est également restreint aux réseaux IP, et ne rend pas compte des pare-feux.

6.1.3.8 TopoMon

TopoMon [61] est un outil de scrutation (*monitoring*, en anglais) pour les réseaux de grilles, qui étend le NWS (*Network Weather Service*, un outil de prévision des performances réseau [161, 150]) avec des informations de topologie réseau. TopoMon utilise la commande `traceroute` pour découvrir la topologie réseau entre les sites surveillés, à la recherche des routes réseau

partagées à travers Internet. Comme TopoMon repose sur NWS, il est restreint aux réseaux IP, et la topologie réseau est décrite lien par lien. TopoMon décrit même les routeurs intermédiaires sur la route d'un site à un autre, afin de détecter les liens réseau partagés. Cette approche ne conduit pas à une description logique des ressources réseau, et elle ne passe pas à l'échelle.

6.1.3.9 Autres travaux

Quelques travaux [46, 67] représentent la topologie d'Internet de manière hiérarchique avec des réseaux longue distance (WAN), des réseaux métropolitains (MAN, *Metropolitan-Area Network*), et des réseaux locaux (LAN). Une grille de calcul ne peut pas être décrite comme le réseau Internet, car elle comprend en plus des réseaux dédiés haute performance (éventuellement non IP), tels que Myrinet ou InfiniBand [196].

Le planificateur de déploiement Sekitei (*cf.* section 5.3.2.2, page 89) décrit les connexions réseau entre les nœuds de calcul une par une, sous la forme d'une grande matrice. Ce modèle de description ne tient compte ni des pare-feux, ni des réseaux non IP, ni de la traduction d'adresses.

Enfin, Topology-d [129] calcule automatiquement les topologies réseau *logiques* à partir de la description des liens physiques entre chaque paire de machines. Cependant, Topology-d est restreint aux réseaux IP, et ne supporte pas les pare-feux.

6.1.4 Discussion

La question de la description des réseaux n'est pas nouvelle. Cependant, avec l'avènement des grilles de calcul, de nouveaux défis sont apparus : les topologies réseau des grilles de calcul ne sont plus limitées à la topologie du réseau Internet. Par exemple, il faut tenir compte de pare-feux, de réseaux non IP, de traduction d'adresses, de réseaux non-hiérarchiques, *etc.* De plus, la description de la topologie réseau doit passer à l'échelle, car les grilles de calcul comportent un grand nombre de ressources. En particulier, la description des ressources réseau ne peut pas décrire chaque lien réseau entre chaque paire de machines : un cluster de n machines donnerait lieu à la description de $\frac{n(n-1)}{2}$ liens réseau, soit $O(n^2)$, ce qui ne passe évidemment pas à l'échelle. En plus de ne pas passer à l'échelle, cette information est difficile à maintenir.

Comme nous venons de le montrer, les résultats actuels parviennent à décrire les nœuds de calcul et de stockage, mais ne permettent pas de décrire des *réseaux* de grilles de calcul de manière satisfaisante. Soit ils ne peuvent pas décrire de réseaux non IP ou non-hiérarchiques, soit ils ne peuvent pas rendre compte de pare-feux *entre* les sites, soit ils décrivent tous les liens physiques un par un, et ne présentent donc pas l'information sous une forme synthétique directement et facilement exploitable par le planificateur de déploiement.

Nous analysons la difficulté à décrire la topologie réseau de la manière suivante. Les nœuds de calcul ont des caractéristiques qui leur sont propres, et qui ne dépendent d'aucun autre nœud, telles que le nombre de CPU, la taille mémoire, l'espace disque, *etc.* Il est donc naturel de rassembler toutes ces informations sur le nœud à décrire, et de lancer un service

d'information sur ce nœud pour donner accès à ses caractéristiques. Pour ce qui est de la description de la topologie réseau, un lien réseau n'appartient pas à un nœud précis, puisque le lien est partagé entre plusieurs nœuds qui sont interconnectés. Ainsi, il est moins naturel de rassembler sur un *nœud* précis les caractéristiques d'un *lien*. En effet, il n'y aucune raison pour que les informations concernant un lien réseau soient attachées à un nœud plutôt qu'à un autre. Et pourtant, un système d'information sur les ressources s'exécute forcément sur un nœud de calcul, et jamais sur un lien réseau.

6.2 Vue d'ensemble de notre modèle de description de la topologie réseau

La section précédente a démontré la nécessité de concevoir un modèle de description de topologie réseau, qui soit suffisamment expressif pour rendre compte de pare-feux entre sites, pour décrire des réseaux non IP ou non-hiérarchiques, et qui soit de suffisamment haut niveau et synthétique pour être facilement et directement exploitable par le planificateur de déploiement. Cette section présente notre modèle de description de topologie réseau pour les grilles de calcul, conçu dans le but de satisfaire ces contraintes.

6.2.1 Description de la topologie logique

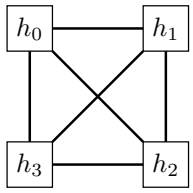
Notre modèle de description de la topologie réseau doit être synthétique pour être le plus facilement exploitable possible par le planificateur de déploiement. Ainsi, nous choisissons de représenter la topologie *logique* plutôt que la topologie *physique* d'interconnexion entre les machines de la grille de calcul. La description de la topologie physique consiste à spécifier chaque câble de connexion, chaque routeur, chaque switch (ou commutateur), chaque hub (ou répéteur), *etc.* La description de la topologie logique indique, de manière plus synthétique, les machines qui peuvent communiquer entre elles, sans préciser les routes et les équipements réseau intermédiaires. Cette approche est adoptée par la plupart des travaux présentés à la section 6.1.3.

Par exemple, les figures 6.2 à 6.4 représentent l'interconnexion via Internet par un seul « lien logique », et non par l'ensemble des chemins réseau et des routeurs par lesquels peuvent passer les messages échangés à travers ce réseau.

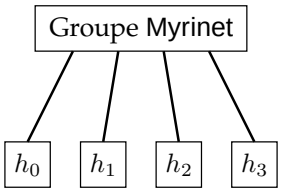
6.2.2 Groupage en sous-réseaux

L'information dont le planificateur de déploiement a besoin est par exemple de savoir que « les quatre machines W, X, Y, et Z appartiennent au même cluster Myrinet », plutôt que de connaître les connexions *individuelles* entre chaque paire de machines. L'information dont le planificateur de déploiement a besoin est une information de haut niveau.

Ainsi, l'idée principale de notre modèle de description de la topologie réseau est de décrire les *groupes de machines qui partagent un sous-réseau commun*, plutôt que de décrire tous les liens logiques un par un. En d'autres termes, notre modèle énumère les machines qui peuvent communiquer entre elles via une technologie réseau, plutôt que de lister, pour chaque machine, toutes les connexions réseau avec d'autres machines.



(a) Énumération des 6 liens logiques entre les machines.



(b) Énumération des 4 machines qui appartiennent au même groupe Myrinet.

FIG. 6.6 – Deux manières de décrire un cluster Myrinet de 4 machines.

Un *groupe réseau* rassemble toutes les machines qui peuvent communiquer entre elles avec des caractéristiques de performance comparables (débit, latence, *etc.*). Par exemple, les n machines d'un cluster Myrinet forment un groupe réseau, et il n'est pas utile de décrire les $\frac{n(n-1)}{2}$ interconnexions entre chaque paire de machines individuellement.

6.2.3 Intérêts du groupage en sous-réseaux

Passage à l'échelle. La figure 6.6(a) illustre la description de chaque lien logique : pour n machines interconnectées, la description spécifie $O(n^2)$ liens. La figure 6.6(b) montre la description du même cluster, avec la spécification de seulement $O(n)$ relations d'appartenance au groupe Myrinet. Le rassemblement des machines en groupes réseau passe mieux à l'échelle que la description de chaque lien logique individuel : notre modèle de description de la topologie donne donc une information plus compacte. Le passage à l'échelle est effectivement une préoccupation importante des grilles de calcul, car elles comprennent un grand nombre de ressources.

Simplification de la tâche du planificateur de déploiement. Les groupes réseau rendent le planificateur plus simple, car la description de la topologie réseau est synthétique, et permet d'obtenir *directement* 4 machines interconnectées par un réseau Myrinet. La description des liens logiques individuels entre chaque paire de machines est une information de trop bas niveau. Notre modèle propose de décrire la topologie réseau à un plus haut niveau, en commençant à reconstituer l'information dont le planificateur peut avoir besoin *dès la description* des ressources. Cette approche évite au planificateur de « *calculer* » l'information de haut niveau dont il a besoin à partir de la description des liens individuels : l'information est déjà « *pré-compilée* » dans la description des ressources. Cependant, nous remarquons que notre modèle de description permet également d'avoir accès à l'information de bas niveau concernant les liens logiques entre chaque paire de machines.

Regroupement de l'information. Comme l'explique la section 6.2.5, des propriétés (performances, *etc.*) peuvent être associées à un groupe réseau. Le regroupement des machines au sein d'un même groupe réseau signifie qu'elles ont sensiblement les mêmes caractéristiques de communication entre elles (débit, latence, *etc.*), et l'information sur ces caractéristiques est rassemblée en un seul endroit, plutôt que d'être disséminée sur la description de chaque lien

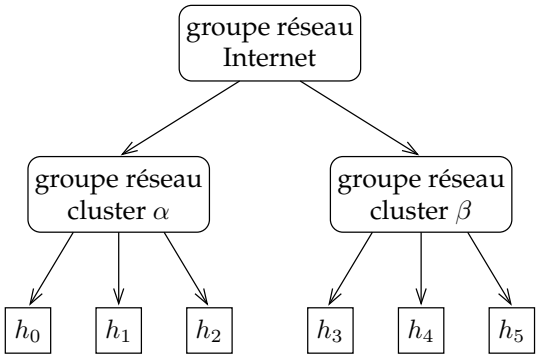


FIG. 6.7 – Graphe représentant la description de la grille A de la figure 6.2.

logique individuel. En outre, le regroupement et l'unification des informations sur les sous-réseaux rend la description des ressources plus facile à maintenir et à mettre à jour.

6.2.4 Graphe de groupes réseau

Une fois le groupage en sous-réseaux acquis, nous décrivons la topologie réseau sous la forme d'un graphe orienté acyclique[4]. Les sommets du graphe sont les groupes réseau présentés à la section 6.2.2 ou bien des ordinateurs : les ordinateurs individuels peuvent être considérés comme des groupes réseau particuliers, constitués d'une seule machine. Les arêtes du graphe sont les relations d'appartenance d'un ordinateur à un groupe réseau ou bien d'un groupe à un autre groupe réseau. Les arêtes sont orientées du groupe réseau parent vers l'ordinateur ou le sous-groupe (ou sous-réseau) qui est inclus dans le groupe parent.

Par exemple, la grille A de la figure 6.2 se décrit par le graphe de la figure 6.7. Les nœuds h_0 à h_2 appartiennent au même sous-réseau FastEthernet, donc ils ont sensiblement les mêmes caractéristiques de communication, et sont les fils du même groupe réseau (cluster α). Les machines des groupes réseau « cluster α » et « cluster β » sont fils du groupe réseau « Internet », puisque leurs ordinateurs peuvent communiquer entre eux via Internet.

Comme l'illustre la grille B de la figure 6.3, une machine peut-être reliée à différents réseaux. La figure 6.8 montre le graphe qui correspond à la grille B : les machines qui appartiennent à plusieurs réseaux sont représentées par des sommets du graphe qui ont plusieurs groupes réseau parents. L'interface réseau qui relie une machine à un groupe réseau est spécifiée au niveau du groupe réseau, puisqu'une machine peut avoir plusieurs interfaces réseau (Ethernet, Myrinet, *etc.*) connectées à différents sous-réseaux. Le groupe réseau fictif appelé « *root vertex* » est un sommet du graphe qui ne sert qu'à *lister* l'ensemble des groupes réseau qui n'ont pas de parent. Les groupes réseau « Myrinet » et « Gigabit Ethernet » représentent des sous-réseaux qui ne sont pas routables depuis Internet, donc ils ne sont pas fils du groupe réseau « Internet » dans le graphe.

Le groupe réseau fictif « *root vertex* » permet de décrire les topologies réseau non-hiérarchiques comme des ensembles de sous-réseaux indépendants les uns des autres. Dans notre modèle de description, la grille D (*cf.* figure 6.5) se représente par le graphe de la figure 6.9.

[4]DAG : *Directed Acyclic Graph*.

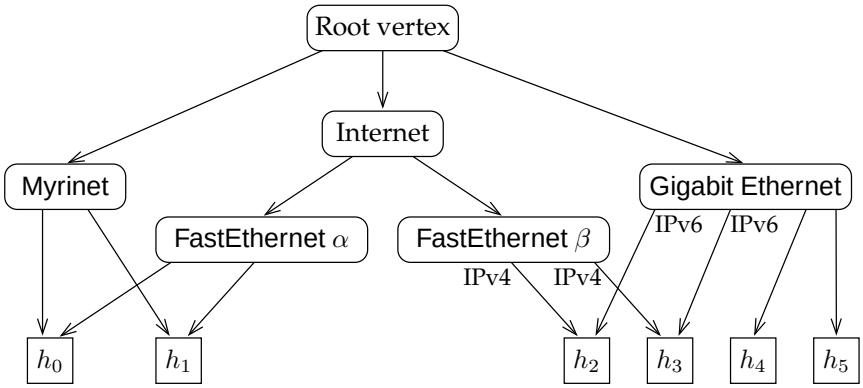


FIG. 6.8 – Graphe représentant la description de la grille B de la figure 6.3 : les machines appartiennent à plusieurs groupes réseau.

Cette grille est constituée de trois sous-réseaux indépendants (« Campus LAN », « Private network » et « Internet »), qui sont représentés par des groupes réseau fils directs de « *root vertex* ». Le pseudo groupe réseau « *root vertex* » n'a pas de parent, et il représente l'ensemble des ressources de la grille. Les machines de deux groupes réseau distincts peuvent communiquer entre elles si et seulement si les deux groupes possèdent un parent (direct ou non) en commun, à l'exception du pseudo groupe « *root vertex* ».

Les sommets du graphe qui n'ont pas de fils sont les nœuds de calcul de la grille. Un nœud de calcul peut être une unique machine accessible par le protocole SSH, ou bien tout un cluster accessible par un système de batch (tel que PBS), ou encore tout un site de calcul accessible par un intergiciel d'accès (tel que Globus). À un nœud de calcul correspond une ou plusieurs méthodes de soumissions de tâches qui permettent d'accéder à l'ensemble des ordinateurs du nœud.

6.2.5 Description des propriétés des réseaux

Les propriétés des nœuds de calcul et des groupes réseau sont associées aux sommets du graphe de description, en tant qu'attributs. Pour les nœuds de calcul, ces propriétés sont la nature et la version du système d'exploitation, le nombre de processeurs, leurs fréquences, leurs architectures matérielles, les logiciels installés, les répertoires de données locaux, *etc.* Pour les groupes réseau, les propriétés décrivent les systèmes de fichiers partagés, les caractéristiques des communications réseau (débit, latence, gigue, taux de perte, *etc.*), les pilotes de cartes réseau (BIP, MX, ou GM pour un réseau Myrinet, par exemple), les protocoles de transport, les ports ouverts et fermés, *etc.* Si un groupe réseau ne définit pas ses propriétés, alors elles sont héritées de son groupe réseau parent.

Les propriétés sont définies *globalement* pour tous les liens à l'intérieur d'un groupe réseau, et non *individuellement* pour chaque lien logique. Cette approche est raisonnable, dans le sens où les caractéristiques de performances des liens réseau sont sensiblement uniformes au sein d'un sous-réseau (réseau local, cluster, *etc.*), et le planificateur de déploiement n'a pas besoin d'une extrême précision pour les valeurs de débit, latence, *etc.*, mais seulement d'ordres de grandeur. De plus, cette approche permet de définir un moins grand nombre de propriétés

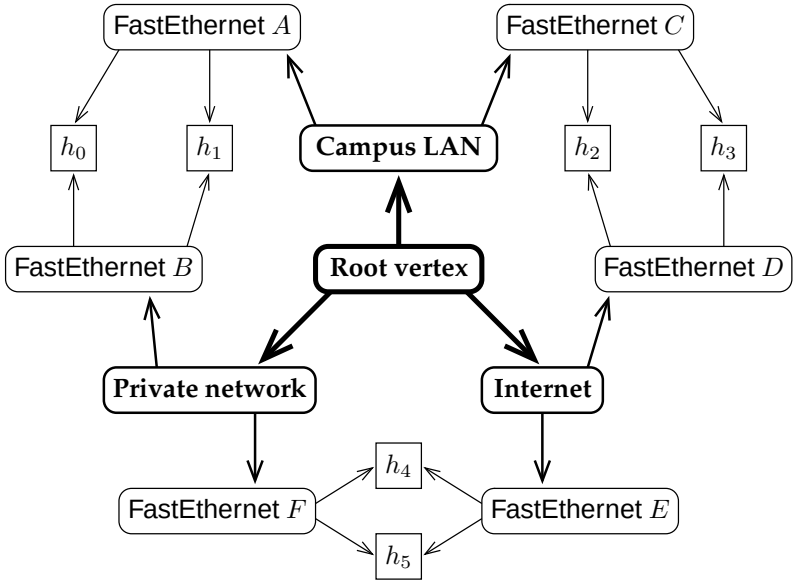


FIG. 6.9 – Graphe représentant la description de la grille D, composée de sous-réseaux suivant une topologie non-hiérarchique.

réseau (une par groupe au lieu d'une par lien réseau), et elle simplifie la maintenance et la mise à jour des informations. Enfin, la notion de propriétés attachées à un groupe réseau permet de facilement définir des groupes de diffusion *multicast* entre plusieurs machines.

6.2.6 Pare-feux, NAT, liens asymétriques

Parmi les propriétés associées aux groupes réseau, notre modèle permet de décrire les caractéristiques de performances des réseaux, mais également les filtrages des pare-feux, les mécanismes de traduction d'adresses (NAT) et les liens asymétriques. En effet, les propriétés d'un groupe réseau peuvent être spécifiées en fonction de la provenance et/ou de la destination des communications.

La figure 6.4, où les communications réseau sont restreintes par des pare-feux, est décrite suivant notre modèle comme l'illustre la figure 6.10. Le groupe parent « Internet » possède trois fils, à savoir les groupes réseau « Domaine A » à « Domaine C ». Les propriétés du groupe réseau « Internet » sont spécifiées avec des sources et des destinations bien précises, référencées parmi les groupes réseau fils. Si la provenance ou la destination d'une propriété d'un groupe réseau n'est pas précisée, alors cette propriété s'applique quelle que soit la source ou la destination, respectivement, des communications. Les règles énoncées dans les propriétés d'un groupe réseau s'ajoutent à celles qui affectent les groupes réseau parents. Une propriété énoncée sans source ni destination, donc valable pour toutes les communications entre les groupes réseau fils, est moins spécifique qu'une propriété exprimée avec une source ou une destination précise, qui est elle-même moins spécifique qu'une propriété énoncée avec à la fois une source et une destination fixées. Les règles les plus spécifiques ont une précédence supérieure à celle des règles moins spécifiques. Ainsi, sur la figure 6.10, la règle non spécifique qui précise que tous les ports sont ouverts (depuis toutes les sources et vers toutes les des-

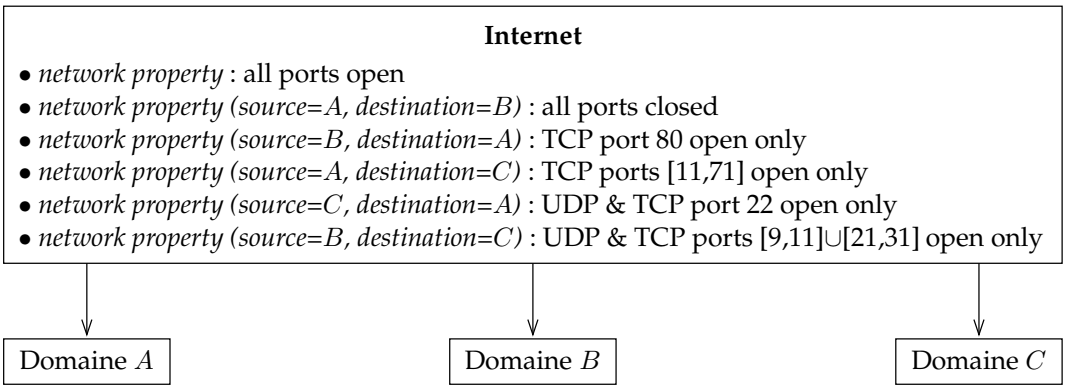


FIG. 6.10 – Graphe représentant la description de la grille C, dont les communications sont restreintes par des pare-feux.

tinations) est une règle générale, qui est modifiée par les règles plus spécifiques qui suivent. Dans cet exemple, les 5 dernières propriétés l'emportent, du point de vue de la précédence, sur la première propriété, puisqu'elles sont plus spécifique. Notons qu'il est inutile de préciser que les communications du « Domaine C » vers le « Domaine B » ne sont limitées par aucun pare-feu, puisque c'est la règle générale, exprimée par la première propriété.

Pour rendre compte des mécanismes de traduction d'adresses (NAT), les propriétés d'un groupe peuvent spécifier les adresses IP d'une ou plusieurs passerelles de traduction. Pour décrire les sous-réseau en adresses IP privées, les propriétés d'un groupe réseau peuvent spécifier une ou plusieurs passerelle d'accès (et les méthodes d'accès à ces nœuds, telles que SSH, Globus GRAM, *etc.*).

La possibilité de spécifier la source et/ou la destination des propriétés entre les groupes réseau permet également de décrire des liens aux performances asymétriques. Les propriétés de communication entre les machines d'un groupe réseau (ou entre les sous-réseau d'un groupe) sont associées au groupe réseau, tandis que les propriétés de communication d'un groupe réseau avec un autre groupe sont attachées au groupe réseau parent qu'ils ont en commun. La détection des incohérences dans les propriétés d'un groupe réseau est facilitée, car toutes les propriétés concernant les relations entre des groupes réseau sont rassemblées en un seul et unique endroit, à savoir leur groupe réseau parent.

6.2.7 Diversité des méthodes de soumission de tâches

Comme nous l'avons vu à la section 6.1.1, il peut exister plusieurs granularités pour soumettre des tâches sur des ressources de calcul : SSH peut être utilisé pour lancer un programme sur une machine individuelle d'un cluster, et le système de batch PBS pour soumettre une tâche globalement sur plusieurs machines du même cluster. La description des ressources doit spécifier toutes les possibilités de soumission de tâches, sans laisser penser qu'il s'agit de ressources distinctes.

Pour ce faire, notre modèle de description introduit la notion de « groupes de ressources équivalentes » (ERG, *Equivalent Ressources Group*). Un ERG rassemble des machines et/ou des

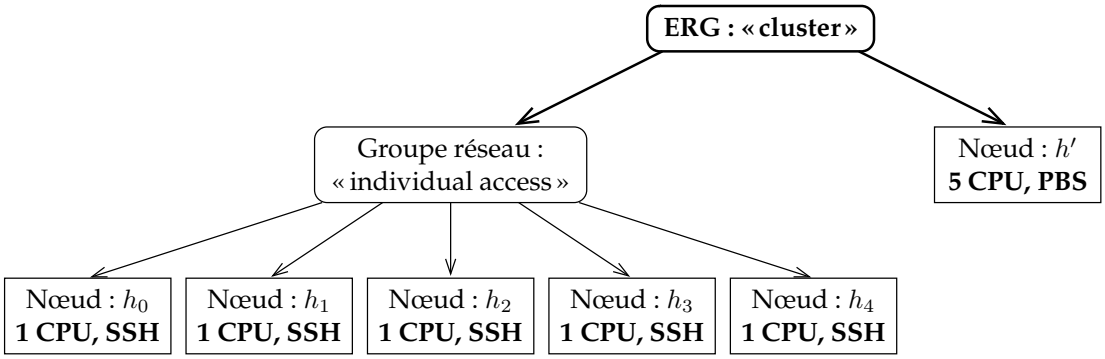


FIG. 6.11 – Graphe représentant la description de la grille E, constituée d'un cluster dont les machines sont accessibles individuellement par SSH ou globalement par PBS.

groupes réseau qui présentent différentes granularités de méthodes de soumission de tâches, mais qui représentent les mêmes ensembles de ressources.

Par exemple, la figure 6.11 illustre la description suivant notre modèle de la grille E (*cf.* figure 6.1). Le groupe réseau « individual access » représente le cluster où chaque machine individuelle est accessible par SSH, et le nœud de calcul h' représente le cluster où les 5 processeurs des machines sont accessibles par PBS. L'inclusion de ce groupe réseau et de ce nœud dans le groupe ERG « cluster » signifie que ces deux descriptions sont deux vues différentes des mêmes ressources de calcul, avec des méthodes d'accès distinctes.

6.2.8 Qui produit la description des ressources réseau ?

Les sections précédentes ont présenté notre modèle de description des ressources, et en particulier de la topologie réseau des grilles de calcul, mais nous n'avons pas encore évoqué la question de savoir comment le descripteur des ressources est *généré*. Notre modèle n'impose rien sur ce point : la description des ressources peut être *produite automatiquement* par un programme, ou bien *écrite à la main* par un administrateur de site.

TopoMon (*cf.* section 6.1.3.8) ou d'autres outils [40, 54] pourraient servir à générer une telle description de la topologie réseau, en ce qui concerne les réseaux IP. Cependant, le modèle de description comprend des informations qui ne sont pas facilement calculables ou détectables automatiquement par un programme : une intervention humaine est nécessaire, par exemple pour spécifier les passerelles d'accès aux réseaux sous adresses IP privées, *etc.* Mais la description des ressources se fait une fois pour toutes, car elle peut être utilisée pour le déploiement de différents types d'applications.

Enfin, les résultats du groupe de travail GMA (*Grid Monitoring Architecture*, [157]) du GGF pourraient permettre d'aider à la production de la description des ressources réseau : le GMA s'intéresse à la surveillance (*monitoring*, en anglais) des performances des éléments distribués d'une grille de calcul, dont les réseaux.

6.2.9 Discussion

Cette section a présenté notre modèle de description de la topologie réseau des grilles de calcul. Pour faciliter son exploitation par le planificateur de déploiement, nous avons opté pour une description d'assez haut niveau : ce modèle décrit la topologie logique d'interconnexion des machines (et non les liens physiques réels), et il rassemble au sein de groupes réseau des machines qui sont interconnectées par un même type de réseau.

La différence fondamentale entre les travaux apparentés et notre modèle de description de la topologie réseau porte sur la *nature des objets* que nous associons aux sommets et aux arêtes du graphe. Les travaux apparentés projettent les *ordinateurs* sur les *sommets* du graphe, et les *liens réseau* (physiques ou logiques) sur ses *arêtes*, tandis que notre modèle de description projette les *groupes réseau* sur les *sommets*, et les *relations d'inclusion* entre groupes réseau sur les *arêtes* du graphe. Notre modèle de description de la topologie réseau passe mieux à l'échelle, car il génère moins d'arêtes ($O(n)$) que les travaux apparentés ($O(n^2)$). En effet, les liens réseau logiques au sein d'un groupe réseau sont *implicites* dans notre modèle.

Enfin, notre modèle de description spécifie l'organisation générale de l'information sur la topologie réseau, mais n'entre pas dans les détails de la façon dont les caractéristiques de performance des réseaux peuvent être décrites. Les résultats du groupe de travail NMWG du GGF (*cf.* section 6.1.2.2) peuvent être intégrés à notre modèle pour décrire les caractéristiques de performance des réseaux parmi les propriétés des groupes réseau.

6.3 Spécification de notre modèle de description des ressources

La section précédente a présenté une vue d'ensemble de notre modèle de description de la topologie réseau des grilles de calcul. Cette section spécifie un formalisme qui permet de représenter l'information sur la description des ressources conformément à notre modèle.

6.3.1 Organisation générale

Comme l'illustre la figure 6.12, la description des ressources d'une grille suivant notre modèle est constituée d'une liste de nœuds de calcul `compute_nodes`, d'une liste de propriétés `system_properties` qui peuvent être associées aux nœuds de calcul, d'une liste de groupes réseau `res_groups` (ou plus généralement une liste de groupes de *ressources*), et une liste de propriétés `group_properties` qui peuvent être associées aux groupes de ressources. Tous ces éléments sont détaillés dans les sections suivantes.

Cette spécification de notre modèle intègre la possibilité que la description des ressources soit *distribuée*, grâce à l'élément `remote_res_descriptors` : toutes les ressources ne sont pas forcément décrites dans un seul et unique descripteur. Un document de description de ressources peut référencer un autre document qui décrit d'autres ressources. L'élément `id` permet de référencer le descripteur de ressources qui se trouve à l'endroit désigné par l'élément `location`. Ce dernier contient un schéma (**`http://...`**, **`ldap://...`**, *etc.*) qui indique par quelle méthode le descripteur peut être récupéré.

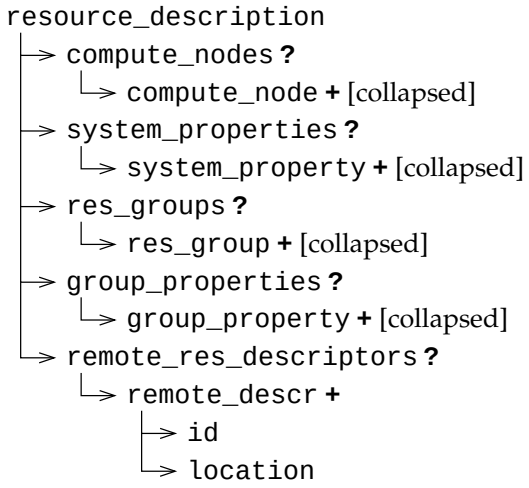


FIG. 6.12 – Hiérarchie du modèle de description des ressources.

6.3.2 Nœuds de calcul

Comme l'illustre la figure 6.13, un nœud de calcul possède un identificateur `id`, utile pour le référencer dans des groupes réseau. Un nœud de calcul possède également une ou plusieurs méthodes de soumission de tâche à distance : plusieurs protocoles sont envisageables, tels que SSH, RSH, Globus GRAM, *etc.* (cette spécification est extensible). Chaque méthode de soumission possède un identificateur, qui permettra au plan de déploiement d'identifier celle qu'il a choisie.

Pour le protocole Globus2, l'élément `contact_string` comprend le nom de la machine sur laquelle le daemon *GateKeeper* (*cf.* section 3.3.2.1, page 46) attend les requêtes, son numéro de port TCP, et d'autres informations spécifiques à Globus. L'élément `vendor_mpi` indique si le *GateKeeper* de Globus est capable de lancer des programmes MPI sur les ressources qu'il gère (`vendor_mpi` précise l'implémentation de MPI avec laquelle Globus a été installé, telle que MPICH-GM par exemple).

L'élément `sys_prop_ref_id` permet d'associer un ensemble de propriétés à ce nœud de calcul (*cf.* section suivante), en spécifiant l'identifiant de cet ensemble de propriétés. Si ces propriétés sont décrites dans un autre descripteur, l'élément `remote_descr_ref_id` référence un élément `remote_descr` qui indique où trouver le descripteur qui contient les propriétés du nœud de calcul.

Enfin, l'élément `network_interface` peut déclarer une ou plusieurs interfaces réseau attachées à ce nœud de calcul : c'est utile pour spécifier une interface par défaut si le nœud de calcul est une machine, et non un cluster de plusieurs machines gérées par un système de batch ou un intergiciel d'accès. Si la machine possède plusieurs interfaces réseau reliées à différents sous-réseaux, alors nous verrons à la section 6.3.4 que les interfaces réseau doivent être définies au moment de l'inclusion d'une machine dans un groupe réseau. Le type de l'interface peut être IPv4, IPv6, HostName_v4, HostName_v6, Myrinet, *etc.* Le nom `name` de l'interface est, selon le type, une adresse IP ou un nom de machine (en version IPv4 ou IPv6), un numéro de carte Myrinet, *etc.*

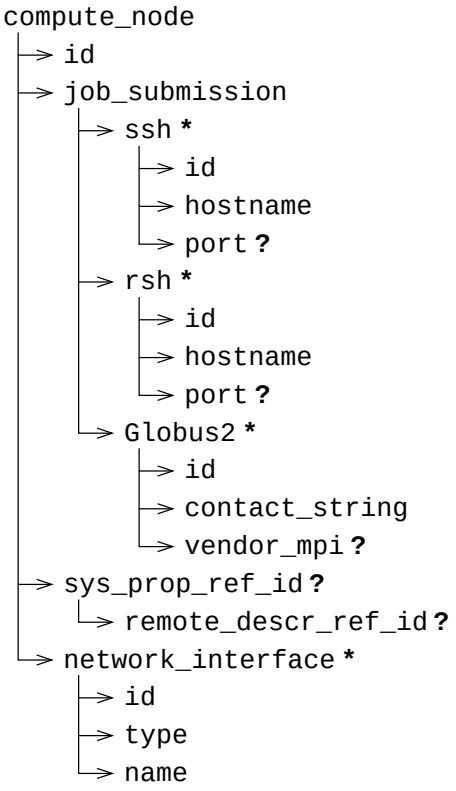


FIG. 6.13 – Description des nœuds de calcul dans notre modèle.

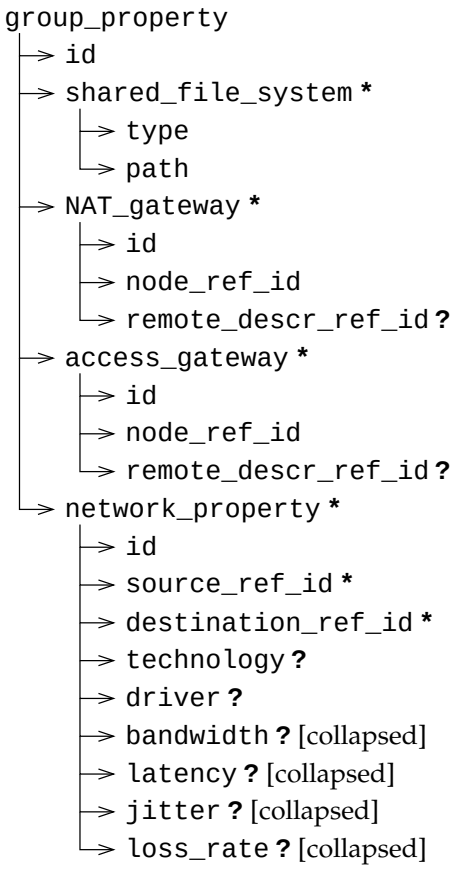


FIG. 6.14 – Description des propriétés des groupes réseau dans notre modèle.

```
system_property
├─> id
├─> operating_system ?
│   ├─> OS_name
│   ├─> OS_bits *
│   └─> OS_version ?
├─> computer_ISA ?
│   ├─> ISA_name
│   └─> ISA_version ?
├─> node_count ?
├─> CPU_count ?
├─> CPU_speed_MHz ?
├─> total_RAM_size_MB ?
├─> environment_variable *
│   ├─> (pre_separator|post_separator) ?
│   ├─> var_name
│   └─> var_value
├─> installed_software *
│   ├─> soft_name
│   ├─> soft_version ?
│   └─> path *
├─> local_scratch_dir *
│   ├─> type ?
│   └─> path
└─> external_information ?
    └─> ( Globus_MDS2 | Unicore | ... )
```

FIG. 6.15 – Description des propriétés des nœuds de calcul dans notre modèle.

6.3.3 Propriétés des nœuds de calcul

Comme l'illustre la figure 6.15, un ensemble de propriétés d'un nœud de calcul possède un unique identificateur qui permet d'attacher cet ensemble de propriétés à un ou plusieurs nœuds de calcul.

Ces propriétés peuvent spécifier le système d'exploitation (élément `operating_system`) par son nom, son format d'exécutable (32 bits et/ou 64 bits), sa version. Les propriétés peuvent préciser l'architecture matérielle avec l'élément `computer_ISA`[5]. Comme mentionné à la section 6.2.4, un nœud de calcul peut être une unique machine, tout un cluster, ou tout un site de calcul d'un institut. Ainsi, l'élément `node_count` permet de préciser le nombre de machines qui appartiennent au nœud de calcul ; `CPU_count` donne le nombre total de processeurs du nœud de calcul, `CPU_speed_MHz` donne la fréquence des processeurs, et `total_RAM_size_MB` spécifie la taille totale de mémoire disponible sur les ordinateurs du nœud de calcul.

L'élément `environment_variable` permet de préciser les variables d'environnement qui doivent être définies (avec leurs valeurs) pour pouvoir exécuter des programmes sur le

[5]ISA : *Instruction Set Architecture*.

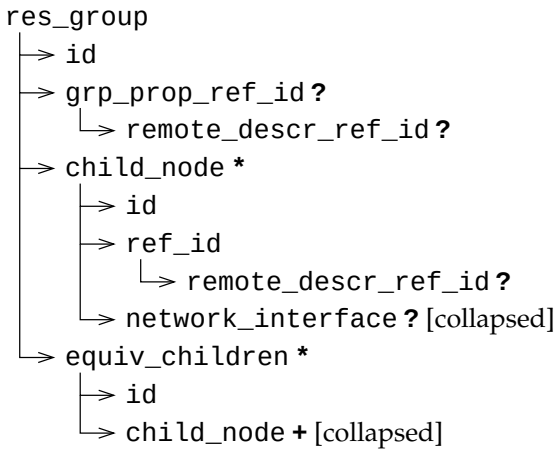


FIG. 6.16 – Description des groupes réseau dans notre modèle.

nœud de calcul. Si le séparateur `pre_separator` ou `post_separator` est absent et que la variable est déjà définie dans l'environnement courant, alors la variable est écrasée. Si un séparateur est défini, alors la valeur est concaténée (préfixée avec `pre_separator` ou suffixée avec `post_separator`) avec la valeur courante de la variable, en utilisant le séparateur indiqué.

L'élément `installed_software` permet de préciser les logiciels particuliers qui sont installés, quelles sont leurs versions, et où ils sont installés (avec l'élément `soft_path`) sur les machines. L'élément `local_scratch_dir` permet de lister les répertoires locaux dans lesquels tout utilisateur peut lire et écrire des fichiers, en précisant leurs chemins d'accès `path`, et éventuellement le type du système de fichiers (ext3, NTFS, FAT32, *etc.*).

Enfin, l'élément `external_information` permet de compléter cette description des propriétés d'un nœud de calcul, en allant lire les informations détenues par le système d'information MDS de Globus, celui d'UNICORE, *etc.* dont les coordonnées sont spécifiées.

6.3.4 Groupes réseau

Les deux sections précédentes ont décrit les nœuds de calcul, sans faire intervenir notre modèle de description de la topologie réseau. Maintenant, nous présentons la spécification qui permet de décrire la topologie réseau conformément à notre modèle.

Comme l'illustre la figure 6.16, un groupe réseau `res_group` (ou plus généralement « groupe de ressources ») possède un identifiant `id`, car il peut être lui-même inclus dans un autre groupe réseau, donc il doit pouvoir être référencé. L'élément `grp_prop_ref_id` contient un identificateur qui permet d'associer des propriétés au groupe réseau (*cf.* section suivante). Si les propriétés du groupe réseau sont définies dans un autre descripteur de ressources, alors l'élément `remote_descr_ref_id` donne l'identificateur de l'élément `remote_descr` qui indique où trouver le descripteur qui contient les propriétés du groupe réseau.

Un groupe réseau parent possède une liste de groupes réseau fils `child_node` qui appartiennent au groupe réseau parent. Les groupes réseau fils sont des sous-réseaux ou des

machines inclus dans le groupe réseau parent. Un groupe réseau fils possède un identificateur, utile pour décrire des propriétés de groupe réseau qui sont asymétriques (pare-feux, performances, *etc.*). Il référence un groupe réseau fils ou bien un nœud de calcul par son identificateur grâce à l'élément `ref_id`. Si le nœud de calcul ou le groupe réseau fils n'est pas défini dans le même descripteur de ressources, alors `remote_descr_ref_id` pointe vers un élément `remote_descr` qui indique où trouver le descripteur qui contient le nœud de calcul ou le groupe réseau fils. Si le fils du groupe réseau est un nœud de calcul, l'élément `network_interface`, dont le contenu est spécifié sur la figure 6.13, permet de préciser par quelle interface réseau la machine est reliée au groupe réseau en cours de description.

Enfin, l'élément `equiv_children` permet de spécifier les groupes de ressources équivalentes (ERG, *cf.* section 6.2.7). Il possède un identificateur `id` qui a le même rôle que pour les `child_node`. L'élément `equiv_children` liste plusieurs groupes réseau ou nœuds de calcul (`child_node`) qui représentent les mêmes ressources, mais avec des méthodes de soumission de tâches qui présentent des granularités différentes.

6.3.5 Propriétés des groupes réseau

Comme l'illustre la figure 6.14, un ensemble de propriétés de groupe réseau (élément `group_property`) possède un identificateur, utile pour associer cet ensemble de propriétés à un ou plusieurs groupes réseau. L'élément `shared_file_system` permet de décrire les systèmes de fichiers partagés par les nœuds de calcul et les sous-groupes d'un groupe réseau ; il est caractérisé par un chemin d'accès `path` et par un type (NFS, PVFS, *etc.*).

L'élément `NAT_gateway` permet de spécifier une ou plusieurs passerelles de traduction d'adresses pour toutes les machines qui appartiennent au groupe réseau : `node_ref_id` pointe vers une machine, et `remote_descr_ref_id` permet de localiser le descripteur de ressources dans lequel la machine est décrite. Pour les réseaux sous adresses IP privées, la ou les passerelles d'accès sont spécifiées par `access_gateway` : elles sont décrites de la même manière que les passerelles NAT.

Enfin, les propriétés réseau du groupe réseau sont décrites par `network_property`. De telles propriétés peuvent être définies pour une ou plusieurs sources précises, et/ou une ou plusieurs destinations précises. Les éléments `source_ref_id` et `destination_ref_id` référencent des nœuds de calcul ou des groupes réseau qui sont des fils du groupe réseau en cours de description : ces éléments pointent vers les identificateurs des `child_node` ou `equiv_children` du groupe réseau. La description du débit `bandwidth`, de la latence `latency`, de la gigue `jitter` et du taux de perte `loss_rate` peut mentionner les valeurs minimale, maximale, moyenne, la variance dans le temps, la méthodologie employée pour obtenir le résultat numérique, *etc.* en utilisant les résultats du groupe de travail NMWG du GGF.

6.3.6 Distribution de l'information

De manière intrinsèque, cette spécification permet à l'information d'être distribuée. Un descripteur de ressources est un document qui décrit une *partie* des ressources de la grille de calcul : il est géré (produit et mis à jour) par un administrateur de cluster ou de site, suivant l'étendue des ressources définies dans le descripteur. Aucune granularité de distribution

de l'information n'est imposée par notre spécification. En particulier, les descripteurs de ressources n'ont aucune raison d'être attachés aux ressources qu'ils décrivent. Par exemple, la description d'un cluster n'est pas nécessairement localisée sur le nœud frontal du cluster, et le service d'information MDS2 de Globus ne s'exécute pas obligatoirement sur la même machine que le *GateKeeper* qui sert à lancer les programmes.

Divers protocoles peuvent être utilisés pour rapatrier les descripteurs de ressources, tels que LDAP[6] pour le système d'information MDS2 de Globus, HTTP ou FTP, *etc.* Cette spécification pourrait aussi s'interfacer avec des techniques pair-à-pair de stockage et découverte d'information distribuée [96, 23]. Cependant, les descripteurs de ressources doivent être accessibles (sans pare-feux qui empêchent les communications, par exemple) depuis n'importe quel client sur Internet, depuis lequel l'outil de déploiement peut avoir besoin de rapatrier ces informations.

Enfin, les descripteurs de ressources pourraient être répliqués pour supporter la défaillance des systèmes d'information. Cependant, cette réplication nécessite de maintenir la cohérence entre les différents réplicas, et de référencer les descripteurs par un nom logique sans préciser leurs localisations (techniques pair-à-pair).

6.3.7 Discussion

Suivant cette spécification, notre modèle de description des ressources peut facilement s'intégrer dans tout système d'information capable de stocker des fichiers (MDS2 de Globus, HTTP, FTP, *etc.*). Les informations sur les ressources peuvent être dynamiques : non seulement les descripteurs peuvent être mis à jour automatiquement en fonction de la charge des processeurs, des congestions réseau, *etc.*, mais ils permettent également de compléter leurs informations avec des systèmes d'informations dynamiques tels que le MDS2 de Globus (avec l'élément `external_information`).

Notre modèle de description et notre spécification donnent lieu à des descriptions de grilles de calcul qui passent à l'échelle pour deux raisons :

- les informations peuvent être distribuées de manière naturelle ;
- le modèle de description de la topologie réseau est compact grâce à la notion de groupes réseau.

Enfin, notre spécification est extensible : par exemple, nous pourrions ajouter des champs qui contiennent des informations sur les charges des nœuds de calcul. Il pourrait également être utile de spécifier, en plus des méthodes de soumission de tâches, des méthodes de *transferts de fichiers*. En effet, un nœud de calcul peut accepter différents protocoles de transferts de fichiers : HTTP dans un sens et GridFTP dans l'autre sens. Les nœuds de stockage pourraient alors être traités comme des nœuds de calcul, sans méthode de soumission de tâches, avec zéro `node_count`, mais avec une ou plusieurs méthodes de transfert de fichiers.

6.4 Conclusion

Ce chapitre a présenté les types de grilles que nous devons savoir décrire pour permettre au planificateur de déploiement de faire des choix en connaissance de cause. Les grilles réa-

[6] LDAP : *Lightweight Directory Access Protocol*, un protocole de stockage et recherche d'information stockée à distance sous forme hiérarchique.

listes sont constituées de réseaux haute performance non IP, de réseaux non-hiérarchiques, de sous-réseaux en adresses IP privées, de pare-feux et de passerelles de traduction d'adresses, de liens aux performances asymétriques, *etc.* Nous avons ensuite proposé un modèle et une spécification de ce modèle de description qui sache rendre compte de toutes ces informations, notamment en ce qui concerne la topologie réseau. L'idée principale réside dans la notion de groupe réseau, qui permet de décrire des topologies réseau de façon compacte et synthétique, donc facilement exploitable par la planificateur de déploiement.

Nous remarquons que ce modèle n'est pas seulement utile pour faire de la planification de déploiement : il pourrait également être utile pour générer des topologies réseau artificielles, afin de mener des expériences de simulation. Ce modèle de description de la topologie réseau pourrait permettre de tester des algorithmes (de routage, de communications, de recherche d'informations, de découverte de ressources) dans des réseaux complexes de grandes dimensions par la simulation.

> « *Qui peut le plus, peut le moins* »

Comme le détaille la section 10.3 (page 185), nous avons implémenté notre modèle de description et nous l'avons intégré au service d'information MDS de Globus. Cependant, cette contribution n'est pas spécifique aux grilles de calcul. Les grilles sont des infrastructures complexes, qui comprennent des sous-ensembles plus simples, tels que des clusters. Donc notre modèle de description peut aussi décrire de simples clusters, des sites de calcul, des réseaux locaux, des fédérations de clusters, *etc.*

Enfin, jusqu'ici, nous avons considéré que les ressources à décrire étaient des machines (matériel et système d'exploitation) ou des liens réseau. Cependant, les ressources pourraient très bien être des *services* ou des daemons spécialisés, tels que des serveurs NetSolve. Notre modèle devrait alors être étendu pour rendre compte non plus seulement des ressources matérielles, mais également des ressources logicielles.

Comme le chapitre 5 l'a montré, la description des ressources n'est pas la seule information dont le planificateur de déploiement a besoin : il faut aussi fournir une description de l'application à déployer. Le chapitre suivant présente nos propositions de formalismes pour décrire des applications MPI et GRIDCCM.

Chapitre 7

Descriptions spécifiques d'applications

Sommaire

Le chapitre 5 a montré que l'outil de déploiement automatique a besoin d'une description *spécifique* de l'application à lancer, et que cette description doit être indépendante des infrastructures d'exécution, afin de déployer l'application dans différents environnements sans modifier sa description. Ce chapitre présente nos propositions de formats de descriptions spécifiques d'applications MPI et d'applications GRIDCCM.

7.1 Description spécifique et packaging d'applications MPI

À ce jour, il n'existe pas de formalisme pour décrire des applications MPI. En effet, comme l'a souligné le chapitre 4, l'utilisateur qui veut déployer une application MPI doit rassembler toutes les informations sur l'application, telles que nous les identifions dans la section suivante. La description spécifique d'une application MPI est écrite par le développeur qui a conçu l'application, et non par l'utilisateur qui veut la déployer. Tant que l'application n'est pas modifiée, sa description reste inchangée.

De plus, la description spécifique d'application doit être indépendante de toute infrastructure d'exécution particulière, afin de permettre facilement son déploiement dans d'autres environnements sans modifier la description d'application. Par exemple, le fichier RSL de la

figure 4.4 (page 61) n'est pas une description d'application, car il mêle des informations sur les ressources (noms de machines) et sur l'application (localisation des exécutables) : c'est un langage de soumission de tâches.

7.1.1 Éléments utiles pour décrire les applications MPI

Parmi les éléments évoqués à la section 5.2.1 (page 84), les informations suivantes sont utiles pour décrire une application MPI :

1. la liste des programmes qui constituent l'application ; le plus souvent, une application MPI est constituée d'un seul programme (SPMD), mais il n'est pas exclu qu'elle soit MPMD ;
2. la localisation des implémentations (versions compilées) de ces programmes et les protocoles utilisables pour récupérer ces fichiers ;
3. la nature et la version des systèmes d'exploitation et des architectures matérielles pour lesquels les implémentations disponibles ont été compilées ;
4. les dépendances des programmes vis-à-vis de bibliothèques, fichiers de configuration ou de données, environnement d'exécution, *etc.* dont la version peut être précisée ;
5. les paramètres de configuration et l'environnement (variables, répertoire courant, *etc.*) des instances de programmes ;
6. les arguments à passer en ligne de commande aux instances de programmes.
7. des contraintes sur le nombre d'instances qui devront être déployées pour chaque programme (degré de parallélisme) ;
8. la description des groupes de processus qui seront amenés à communiquer intensivement au sein de communicateurs MPI ;
9. la description des schémas de communication entre les processus par le biais de la topologie virtuelle au sein des communicateurs.

Une application MPI peut être décomposée en groupes de processus appelés « communicateurs » : les opérations collectives (*cf.* section 2.3.2.1, page 25) ont lieu au sein de ces communicateurs. Ainsi se justifie le besoin de décrire ces communicateurs (point 8. ci-dessus) s'il y a de nombreuses communications à l'intérieur des communicateurs, et relativement peu de communications entre eux.

Dans la plupart des applications parallèles, chaque processus d'un communicateur communique seulement avec un sous-ensemble des autres processus du même groupe. Le schéma de communication s'appelle la « topologie virtuelle » de l'application. La norme MPI permet de décrire ces topologies virtuelles, afin de faciliter le nommage (ou l'indexation) des processus. Par exemple, la fonction **`MPI_Cart_create()`** crée une topologie virtuelle cartésienne de dimension quelconque : la figure 7.1 montre un exemple de topologie cartésienne à deux dimensions (4×3). Dans cet exemple, au lieu de référencer les processus MPI par leur rang (*i.e.*, linéairement), le programmeur peut les référencer par leur coordonnées cartésiennes, telles que $(2, 1)$ par exemple. MPI permet aussi de définir des topologies virtuelles quelconques sous la forme de graphes de communication avec la fonction **`MPI_Graph_create()`**. Ainsi, il est utile de pouvoir décrire ces topologies virtuelles dans la description spécifique d'application MPI, afin de placer les processus qui communiquent beaucoup entre eux d'une manière qui améliore la performance d'exécution de l'application.

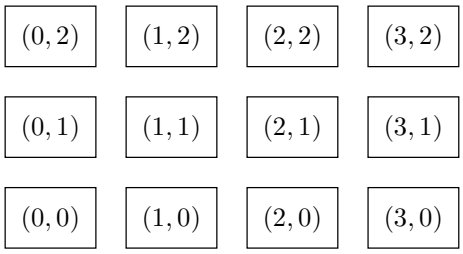


FIG. 7.1 – Exemple de topologie cartésienne MPI à deux dimensions (4×3).

De plus, les implémentations de la bibliothèque MPI ne sont pas intéropérables, car il n'existe pas d'ABI[1] MPI [116]. En effet, chaque implémentation de MPI nomme sa bibliothèque partagée comme elle le souhaite, et les méthodes de lancement des processus MPI varient avec les implémentations. Donc, pour chaque fichier exécutable, il faut indiquer par quelle implémentation de MPI il a été compilé.

7.1.2 Modèle de description spécifique d'applications MPI

Le modèle de description spécifique d'applications MPI que nous proposons s'inspire du format de description des applications CCM (*cf.* section 2.2.2.5, page 16) et du format OSD[2] [158] pour la présentation des informations. Cependant, la *structure* et le *contenu* de la description font partie de notre contribution.

7.1.2.1 Programmes, dépendances et environnement

La figure 7.2 illustre la hiérarchie de notre modèle de description des programmes d'une application MPI. L'élément `application` décrit la structure de l'application et est détaillé dans la section suivante. Un ou plusieurs programmes peuvent constituer une application MPI. Chaque programme possède un identificateur `id` unique qui est utilisé dans la description de la structure de l'application.

Un environnement peut être attaché à chaque programme : l'environnement peut définir des variables d'environnement, un répertoire de travail, des dépendances vis-à-vis de bibliothèques ou d'environnements d'exécution, des fichiers d'entrée et de sortie, ainsi que des arguments en ligne de commande. Les variables d'environnement ont un nom et une valeur, et éventuellement un séparateur : si le séparateur est absent et que la variable est déjà définie dans l'environnement courant, alors la variable est écrasée ; si le séparateur est défini, alors la valeur est concaténée (préfixée avec `pre_separator` ou suffixée avec `post_separator`) avec la valeur courante de la variable en utilisant le séparateur indiqué. Le type du répertoire de travail indique s'il doit être local, partagé (via NFS par exemple), ou bien dans le répertoire d'accueil de l'utilisateur. `path` indique un chemin relatif qui doit être créé par l'outil de déploiement (si le répertoire n'existe pas déjà) et où doit s'exécuter le programme. Les dépendances s'expriment sous la forme de chaînes de caractères conventionnelles : par exemple `libm` indique que la bibliothèque mathématique dynamique est nécessaire. Le ou les fichiers

[1] ABI : *Application Binary Interface*.
[2] OSD : *Open Software Description*.

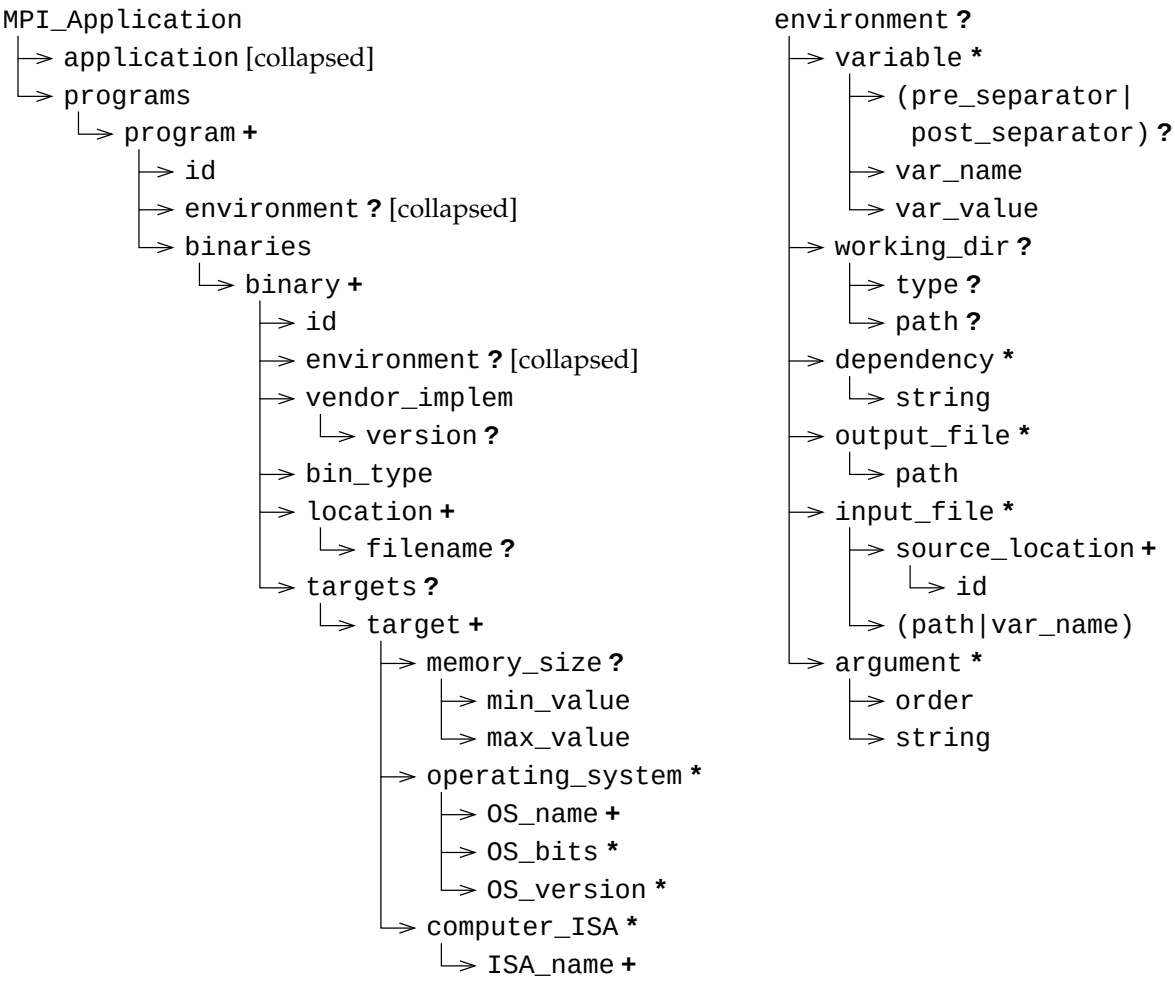


FIG. 7.2 – Hiérarchie du modèle de description spécifique d'application MPI : la description des programmes.

de données nécessaires en entrée `input_file` sont rapatriés depuis l'endroit indiqué par `source_location` (plusieurs sources équivalentes peuvent être mentionnées) et installés dans le répertoire d'exécution, soit sous le nom désigné par `path`, soit sous un nom qui sera stocké dans la variable `var_name`. Le ou les fichiers de sortie `output_file` sont spécifiés, afin que l'outil de déploiement puisse les récupérer et les mettre à la disposition de l'utilisateur après l'exécution. Pour terminer avec l'environnement du programme, une liste d'arguments peut être spécifiée, éventuellement en précisant l'ordre des arguments.

Enfin, `binaries` donne la liste des versions compilées du programme. L'environnement de chaque fichier binaire peut être reprécisé : le répertoire de travail remplace celui qui a déjà été spécifié pour le programme, le cas échéant ; tous les autres éléments de l'environnement (variables, dépendances, fichiers d'entrée ou de sortie, et arguments en ligne de commande) s'ajoutent à ceux déjà définis pour le programme. L'élément `vendor_implem` spécifie l'implémentation de MPI qui a été utilisée pour compiler le programme (MPICH-1, LAM/MPI, PACX-MPI, MPICH-G2, *etc.*), ainsi que sa version : cette information est nécessaire puisqu'il n'existe pas d'ABI pour MPI. L'élément `bin_type` précise le type de fichier binaire (DLL ou exécutable). L'élément `location` spécifie où le fichier binaire peut être récupéré et par quel protocole (URL **`http://...`** ou **`gsiftp://...`**[3] par exemple). Le fichier binaire peut être rapatrié depuis plusieurs endroits différents. Si l'élément `filename` est présent, alors le fichier désigné par `location` est une archive compressée ZIP dont il faut extraire le fichier binaire indiqué par `filename`. Le dernier élément `targets` indique la liste des types de machines avec lesquelles le fichier binaire est compatible ; si cet élément est absent, alors il n'y a aucune contrainte de compatibilité et le fichier binaire est portable sur tout type de machine (*shell script* par exemple). L'élément `target` peut imposer une contrainte sur la taille mémoire, sur le système d'exploitation et sur l'architecture matérielle des machines. Par exemple, le système d'exploitation peut être spécifié par son nom, sa version, et préciser s'il manipule des binaires 32 bits ou 64 bits. Si plusieurs systèmes d'exploitation ou architectures matérielles sont indiqués, alors le binaire est compatible avec chacun de ces systèmes et architectures. Remarquons que plusieurs binaires compilés pour différentes machines (systèmes d'exploitation et architectures matérielles) peuvent être déployés au sein d'une même application pour coopérer, mais que les implémentations de MPI utilisées pour les programmes doivent être identiques, puisque les implémentations de MPI ne sont pas intéropérables.

Dans notre formalisme de description, un fichier binaire d'application MPI peut être un exécutable ou une bibliothèque dynamique (DLL). Habituellement, les applications MPI sont sous la forme d'exécutables. Cependant, PadicoTM [63, 62] permet de lancer des programmes MPI sous la forme de DLL. En effet, PadicoTM est une plate-forme de communication pour la programmation des grilles de calcul : elle permet d'exécuter des applications parallèles, distribuées ou mixtes sans imposer de contrainte de programmation ou de support exécutif particulier. Par exemple, PadicoTM permet d'exécuter des applications utilisant MPI, JAVA, une MVP, CORBA (ainsi que leur combinaison) sur n'importe quel type de réseau, y compris des réseaux à utilisation exclusive tels que Myrinet : PadicoTM se charge d'adapter les exécutifs sur les réseaux, et d'arbitrer l'accès des exécutifs de l'application aux réseaux.

La liste des éléments présentés ici peut être enrichie. Par exemple, l'élément `target` peut spécifier une contrainte sur l'espace disque nécessaire à l'exécution du programme. De plus, nous supposons que les fichiers d'entrée (et les fichiers binaires) ont une localisation bien précise (`source_location`). Or tel n'est pas toujours le cas. Par exemple, Pegasus (*cf.* sec-

[3] Le schéma **`gsiftp`** désigne le protocole GridFTP (*cf.* section 3.3.2.1, page 46).

tion 4.3.3.5, page 67), le RLS de Globus (*Replica Location Service*, *cf.* section 3.3.2.1, page 46) ou les systèmes pair-à-pair permettent de désigner les fichiers par des « noms logiques », et de ensuite de localiser et rapatrier les fichiers à partir de leurs noms logiques. Si un fichier est répliqué en plusieurs endroits, le planificateur peut tenir compte de la localisation des différentes copies du fichier pour placer l'application à proximité des données (comme le fait Pegasus).

La figure 7.3 illustre la description des programmes d'une application MPI. Deux programmes sont disponibles : un « maître » et un « esclave », les deux étant des exécutables compilés avec l'implémentation MPICH-G2. Le premier se récupère soit par FTP, soit par HTTP, et le deuxième par HTTP uniquement. Le programme maître est compilé pour le système d'exploitation AIX, version 4.3, sur PowerPC ; le programme esclave est compilé pour Linux 32 bits sur architecture x86. Le programme esclave doit avoir une variable d'environnement définie (`HAS_A_MASTER=true`), un répertoire de travail bien précis, deux arguments en ligne de commande, un fichier de données en entrée récupérable par FTP, et il dépend de la bibliothèque mathématique.

7.1.2.2 Structure de l'application MPI

La section précédente a expliqué comment notre modèle décrit les différents programmes qui peuvent constituer une application MPI. Nous montrons maintenant comment il décrit quels programmes sont instanciés, et avec quelles cardinalités, dans l'élément `application` qui n'a pas encore été décrit.

Sur la figure 7.4, l'élément `environment` est du même type que dans la section précédente pour la description des programmes : il permet d'enrichir les environnements déjà définis pour les programmes. L'élément `world_size`, facultatif, permet d'imposer des contraintes sur le nombre total de processus MPI de l'application[4]. L'élément `var` permet de définir le symbole qui représente le nombre total de processus de l'application, et l'élément `dummy_int` désigne un entier quelconque. Par exemple, si la variable qui représente le nombre total de processus est n et que `dummy_int` est i, alors contraindre le nombre de processus à un nombre pair s'exprime par $n = 2 \times i$. Pour l'instant, nous n'envisageons que des contraintes simples sur le nombre de processus (puissance de 2, carré parfait, bornes supérieures et inférieures, *etc.*), mais le langage d'expression de ces contraintes peut être étendu.

L'élément `prgrm_instances` donne la liste des programmes à instancier dans l'application : une instance de programme possède un identificateur `id`, dont le rôle est expliqué dans la section suivante, et désigne le programme à instancier par `ref_id`, qui pointe vers l'élément `id` d'un programme `program` décrit dans la section précédente. Une instance de programme peut encore enrichir son environnement avec l'élément `environment`. Enfin, le nombre d'instances du programme à lancer est exprimé dans l'élément `cardinality` : `var` définit le symbole qui représente la cardinalité de cette instance de programme, `dummy_int` désigne un nombre entier quelconque, et le symbole défini par `world_size` représente le nombre total de processus de l'application. Si la cardinalité des processus d'une application MPI n'est pas fixée par la description d'application, alors il appartient au planificateur de déploiement d'en choisir une dans le respect des contraintes de la description spécifique d'application.

[4]Dans un code MPI, le symbole **`MPI_COMM_WORLD`** désigne le communicateur qui comprend *tous* les processus de l'application.

```
<MPI_Application>

  <programs>
    <program id="master_program">
      <binaries vendor_implem="MPICH-G2" bin_type="exec" id="master">
        <location>ftp://ftp.exe-store.org/master.exe</location>
        <location>http://http.exe-store.org/master.exe</location>
        <targets>
          <target>
            <operating_system>
              <OS_name>AIX</OS_name> <OS_version>4.3</OS_version>
            </operating_system>
            <computer_ISA> <ISA_name>PowerPC</ISA_name> </computer_ISA>
          </target>
        </targets>
      </binaries>
    </program>

    <program id="slave_program">
      <environment>
        <variable>
          <var_name>HAS_A_MASTER</var_name> <var_value>true</var_value>
        </variable>
        <working_dir type="local">slave-dir</working_dir>
        <dependency>libm</dependency>
        <input_file>
          <source_location id="spectrum_location">
            ftp://matrix.store.org/spectrum.data
          </source_location>
          <path>input_matrix.data</path>
        </input_file>
        <argument order="10">-in</argument>
        <argument order="11">input_matrix.data</argument>
      </environment>

      <binaries>
        <binary vendor_implem="MPICH-G2" bin_type="exec" id="slave_x86">
          <location>http://x86.store.org/fft.exe</location>
          <targets>
            <target>
              <operating_system>
                <OS_name>Linux</OS_name> <OS_bits>32</OS_bits>
              </operating_system>
              <computer_ISA> <ISA_name>x86</ISA_name> </computer_ISA>
            </target>
          </targets>
        </binary>
      </binaries>
    </program>
  </programs>

<application> ...  </application>

</MPI_Application>
```

FIG. 7.3 – Exemple de description des programmes d'une application MPI au format XML.

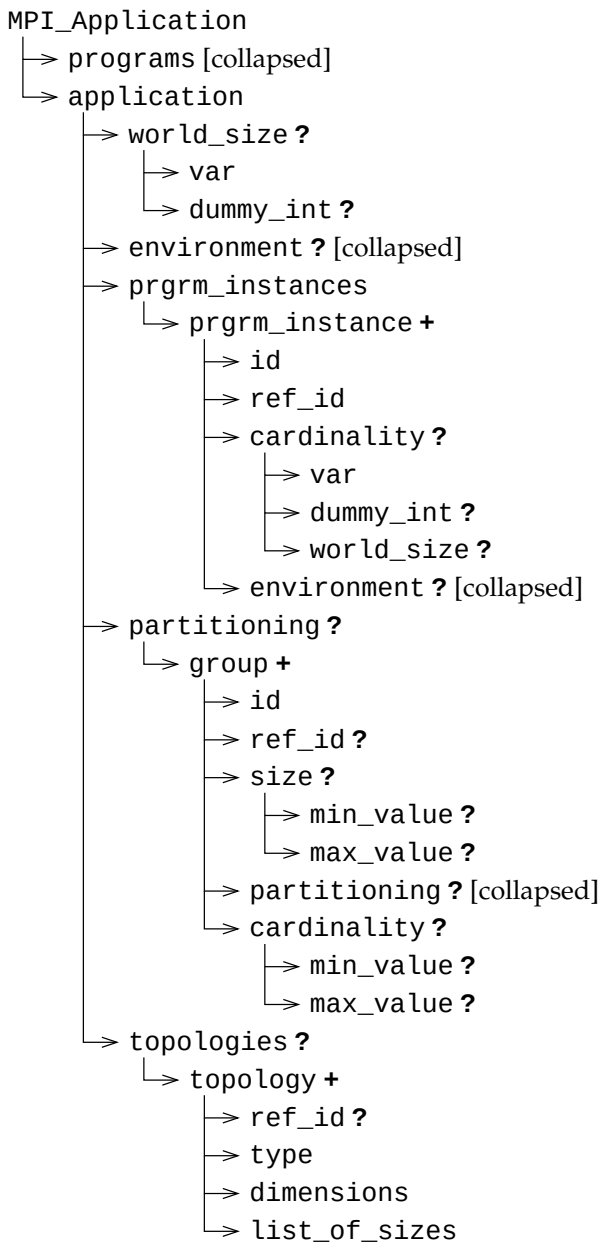


FIG. 7.4 – Hiérarchie du modèle de description spécifique d'application MPI : la structure de l'application.

```
<MPI_Application>
  <programs> ... </programs>
  <application>
    <world_size var="n" dummy_int="i">
      n > 8  &  n = 2**i + 1 &  n < 256*4
    </world_size>
    <environment>
      <argument order="50">-optimize</argument>
    </environment>
    <prgrm_instances>
      <prgrm_instance id="the_master" ref_id="master_program">
        <cardinality var="x"> x = 1 </cardinality>
        <environment>
          <variable pre_separator=":">
            <var_name>PATH</var_name>
            <var_value>/usr/X11/bin</var_value>
          </variable>
        </environment>
      </prgrm_instance>
      <prgrm_instance id="the_slave" ref_id="slave_program"/>
    <prgrm_instances>
    ...
  </application>
</MPI_Application>
```

FIG. 7.5 – Exemple de description de la structure d'une application MPI au format XML.

La figure 7.5 illustre un exemple de description de la structure d'une application MPI, à la suite de la définition des programmes de la figure 7.3. Le nombre total de processus de l'application n'est pas fixé, mais il est contraint : il doit être supérieur à 8, il doit être une puissance entière de 2 plus 1, et il doit être inférieur à 256×4. Dans cet exemple de la figure 7.5, tous les processus de l'application doivent être invoqués avec l'option `-optimize`, et deux programmes doivent être instanciés lors du déploiement : le programme maître sera lancé en un seul exemplaire et avec la variable `PATH` préfixée par `/usr/X11/bin`; le programme esclave sera lancé avec une cardinalité laissée libre, tant que le nombre total de processus de l'application satisfait les contraintes exprimées ci-dessus (via l'élément `world_size`).

L'exploitation et la vérification de la cohérence d'une expression mathématique telle que la contrainte qui définit `world_size` peuvent être réalisées par un moteur d'évaluation d'expressions symboliques fondés sur des algorithmes de Simplex. De tels algorithmes d'analyse d'expressions symboliques et de satisfaction de contraintes (CSP[5]) sont utilisés dans le logiciel Maple[6], par exemple.

[5]CSP : *Constraint Satisfaction Problem*.

[6]Maple est un logiciel de calcul numérique et symbolique.

7.1.2.3 Partitionnement d'une application MPI

Les contraintes décrites précédemment étaient *impératives* : si elles ne sont pas respectées, l'application ne peut tout simplement pas s'exécuter. Par exemple, on ne peut pas exécuter un programme compilé pour Linux sur x86 sur une machine Alpha sous Tru64. Les informations que notre modèle permet de décrire dans cette section et dans la suivante, sur le partitionnement et la topologie virtuelle, sont plutôt *indicatives* : leur respect permet seulement d'obtenir de meilleures performances d'exécution.

L'élément `partitioning` de la figure 7.4 permet de lister récursivement les communicateurs, ou les groupes de processus au sein desquels les communications auront lieu pour la plupart. Cette information permettra au planificateur de déploiement de placer, dans la mesure du possible, les processus qui communiquent intensivement entre eux sur des réseaux haute performance (*cf.* section 7.1.4). L'élément `group` possède un identificateur `id` dont l'utilité est expliquée dans la section suivante. L'élément `ref_id` désigne le programme dont le partitionnement est contraint, en pointant vers l'élément `id` de l'instance de programme `prgrm_instance` décrit dans la section précédente. Si l'élément `ref_id` est absent, alors le partitionnement concerne l'intégralité de l'application MPI. L'élément `ref_id` est interdit pour les sous-groupes, car dans ce cas, il est évident que le partitionnement concerne le groupe parent.

Ce mécanisme n'a de sens que pour les programmes qui sont instanciés plus d'une fois (cardinalité supérieure à 1 processus). Ces programmes peuvent être partitionnés en plusieurs groupes. L'élément `size` permet de borner la taille de ces groupes, et l'élément `cardinality` permet de borner le nombre de groupes.

Enfin, un groupe peut lui-même être partitionné en plusieurs sous-groupes, c'est pourquoi l'élément `group` possède un élément `partitioning` pour contraindre, récursivement, le partitionnement des groupes. Cette possibilité de partitionnement récursif d'une application MPI est raisonnable, dans la mesure où les grilles de calcul peuvent offrir toute une hiérarchie de performances réseau : il peut exister plusieurs ordres de grandeur entre les latences et débits des réseaux longue distance (WAN, Internet), des réseaux locaux (LAN), des réseaux haute performance des clusters (SAN), ou encore par rapport aux vitesses de communication entre les processus d'un même ordinateur à mémoire partagée (SMP[7], CC-NUMA[8], *etc.*).

La figure 7.6 illustre un exemple de partitionnement possible de l'application dont la structure est décrite à la figure 7.5. Ce partitionnement indique qu'il est souhaitable que les instances `the_slave` du programme esclave ne soient pas réparties en plus de quatre groupes de communication haute performance séparés par des liens réseau plus lents. Cette contrainte évite une dispersion trop importante de l'application. Ensuite, chaque groupe peut être partitionné en deux types de sous-groupes : un seul sous-groupe de 2 processus au plus, et un autre sous-groupe qui ne devrait pas être subdivisé en plus de 8 parties. La section 9.2 (page 162) montre que l'implémentation MPICH-G2 sait transmettre ce genre d'information sur le partitionnement à l'application, et qu'elle en tient compte pour l'amélioration des performances des opérations collectives MPI.

Tel qu'il est présenté ici, notre modèle permet de contraindre le partitionnement d'une application MPI de manière *qualitative*. Ce modèle peut être enrichi de contraintes d'ordre

[7] *Cf.* figure 2.5, page 28.
[8] *Cf.* note de bas de page 9, page 44.

```
<MPI_Application>
  <programs> ... </programs>
  <application>
    ...
    <partitioning>
      <group id="slave_group" ref_id="the_slave">
        <cardinality>
          <max_value> 4 </max_value>
        </cardinality>
        <partitioning>
          <group id="slave_subgroup1">
            <size>
              <max_value> 2 </max_value>
            </size>
            <cardinality>
              <min_value> 1 </min_value>
              <max_value> 1 </max_value>
            </cardinality>
          </group>
          <group id="slave_subgroup2">
            <cardinality>
              <max_value> 8 </max_value>
            </cardinality>
          </group>
        </partitioning>
      </group>
    </partitioning>
    <topologies>
      <topology ref_id="slave_subgroup2" type="cartesian">
        <dimensions> 2 </dimensions>
        <list_of_sizes> 4,3 </list_of_sizes>
      </topology>
    </topologies>
  </application>
</MPI_Application>
```

FIG. 7.6 – Exemple de description du partitionnement et de la topologie d'une application MPI au format XML.

quantitatif, par exemple en spécifiant les rapports de débit et/ou de latence entre les processus d'un groupe et ceux localisés dans des groupes différents.

7.1.2.4 Topologie et schémas de communication d'une application MPI

Pour indiquer les schémas de communication au sein des communicateurs MPI, notre modèle de description permet de spécifier les topologies virtuelles de la catégorie des graphes cartésiens. Il pourrait également être utile de représenter les graphes quelconques, mais ces derniers sont nettement moins utilisés dans la programmation MPI.

L'élément `topologies` de la figure 7.4 liste les différentes topologies virtuelles de chaque groupe de processus, afin de permettre au planificateur de déploiement de placer les processus qui communiquent le plus entre eux à proximité[9] les uns des autres (*cf.* section 7.1.4). L'élément `ref_id` permet de spécifier le groupe de processus pour lequel la topologie virtuelle est définie : `ref_id` désigne l'un des identificateurs `id` des groupes présentés dans la section précédente. L'élément `ref_id` est facultatif : s'il n'est pas précisé, alors la topologie virtuelle est définie pour le communicateur **`MPI_COMM_WORLD`**, qui regroupe l'ensemble des processus de l'application MPI. L'élément `type` précise s'il s'agit d'une topologie cartésienne ou bien d'un graphe générique. Pour les graphes cartésiens, `dimensions` spécifie le nombre de dimensions du graphe, et l'élément `list_of_sizes` liste le nombre de processus suivant chacune des dimensions du graphe.

La figure 7.6 illustre comment la topologie virtuelle du groupe de processus d'identificateur `slave_subgroup2` peut être décrite. Ici, l'exemple décrit une topologie cartésienne de dimension 2 comme illustrée à la figure 7.1 (page 121).

7.1.3 Packaging d'applications MPI

Une application peut être packagée pour être distribuée à ses utilisateurs. Pour packager les applications MPI, nous avons choisi d'imiter la technologie de packaging des applications CCM. Un package d'application MPI est une archive compressée au format ZIP, qui contient au moins le fichier XML de description de l'application suivant le modèle présenté dans la section précédente. Dans l'archive ZIP, ce fichier XML de description de l'application doit se trouver dans un répertoire **`meta-inf`**, et avoir une extension **`.mpi`**. Les différents fichiers binaires de l'application peuvent être localisés soit dans l'archive ZIP, soit en dehors de l'archive (sur un site web, un site FTP, *etc.*). C'est la description de l'application MPI qui indique où les fichiers binaires peuvent être récupérés, et par quel protocole.

L'intérêt de pouvoir, dans un package d'application, référencer des fichiers extérieurs est multiple :

- la mise à jour des fichiers binaires est plus simple, puisqu'il n'est pas utile de recréer toutes les archives compressées qui contiennent le binaire à mettre à jour ;
- si le package est réduit à la seule description de l'application et que tous les binaires sont à l'extérieur, alors le package est plus léger à télécharger, et seuls les fichiers binaires réellement utiles seront rapatriés ;
- les fichiers binaires peuvent être partagés plus facilement s'ils entrent dans la compositions de différentes applications.

[9] Proximité au sens de la rapidité des communications réseau.

7.1.4 Planification du déploiement d'applications MPI

La description d'une application MPI laisse de nombreux degrés de liberté quant à la détermination du nombre de processus (s'il n'est pas fixé par les paramètres de contrôle de l'utilisateur), le placement des processus et leur partitionnement. C'est le planificateur de déploiement qui est responsable de faire ces choix, et, en général, il existe plusieurs plans de déploiement qui satisfont les contraintes. Dans cette section, nous montrons que ces questions font l'objet de recherches actives, en citant quelques travaux qui s'attachent à résoudre ces problèmes.

7.1.4.1 Détermination du nombre de processus

Comme annoncé à la section 5.3.2.1 (page 88), le planificateur de déploiement peut être amené à déterminer le nombre de processus d'une application parallèle et le nombre d'instances de chaque programme à lancer.

La communauté de l'ordonnancement produit des résultats encore théoriques [123, 69] qui permettent de déterminer le nombre de processus d'une application parallèle MPI. Par exemple, les résultats de [123] tiennent compte de la topologie et des caractéristiques du réseau pour placer les processus MPI. Ces travaux se classent dans la catégorie de l'ordonnancement de tâches parallèles : ils visent à minimiser les temps d'exécution des applications sur des infrastructures distribuées [69]. Ils mériteraient d'être étendus pour les transformer en algorithmes de planification, avec sélection des ressources et des implémentation, dans le respect des contraintes de compatibilité des systèmes d'exploitation et des architectures matérielles.

7.1.4.2 Respect des informations sur le partitionnement et sur les topologies virtuelles

Des travaux tels que [125, 131] se sont déjà intéressés à la projection des topologies virtuelles MPI sur des infrastructures d'exécution. [125] se place dans le cadre de clusters homogènes avec des topologies réseau irrégulières. Ce travail propose deux algorithmes de placement d'applications MPI en respectant des contraintes sur les topologies virtuelles : l'un optimise le coût global des communications, et l'autre équilibre la charge de communication sur les liens réseau. [131] place aussi les processus d'applications MPI en fonction de leurs communications, mais sur des fédérations de clusters (topologie réseau hiérarchique).

7.1.5 Discussion

Cette section a présenté notre modèle de description spécifique d'applications MPI, tel qu'il a été mis en œuvre dans notre outil de déploiement ADAGE, présenté au chapitre 10. La description est indépendante des ressources d'exécution, et elle est écrite par le développeur une fois pour toutes. Ce modèle permet à l'utilisateur de déployer l'application plusieurs fois sur différentes infrastructures d'exécution, sans avoir à modifier la description de l'application. En particulier, ce modèle de description n'est pas lié aux grilles de calcul.

Notre modèle de description spécifique peut être étendu. Par exemple, il peut être utile de ne pas spécifier explicitement la localisation exacte des fichiers (binaires, données, *etc.*), mais se contenter de les référencer par un *nom logique*. Un système pair-à-pair serait ensuite capable de localiser de façon transparente un exemplaire du fichier. Il peut également être utile d'ajouter

des *contraintes quantitatives* à la description du partitionnement de l'application et de ses topologies virtuelles. Ces contraintes pourraient porter, par exemple, sur les rapports de latences ou de débits réseau entre les processus MPI. Cependant, la conception d'un planificateur qui sache tenir compte de toutes ces contraintes est particulièrement ardue.

Enfin, des travaux sur le placement des processus d'une application parallèle MPI existent, mais ils sont encore relativement théoriques. Ils gagneraient à :

- mêler les algorithmes de détermination du nombre de processus, avec ceux de partitionnement et ceux de projection des topologies virtuelles sur les infrastructures d'exécution ;
- et à être étendus à des algorithmes de planification avec sélection des ressources et des implémentations, et non plus seulement des algorithmes d'ordonnancement de tâches.

7.2 Description spécifique et packaging d'applications GRIDCCM

De même que pour les applications MPI, il n'existe pas, à ce jour, de formalisme pour décrire des applications GRIDCCM. L'utilisateur qui veut déployer une application GRIDCCM doit rassembler toutes les informations utiles au déploiement de l'application. La description spécifique d'une application GRIDCCM est écrite par le développeur qui a conçu l'application, et non par l'utilisateur qui veut la déployer. Tant que l'application n'est pas modifiée, sa description reste inchangée. De plus, la description spécifique d'application doit être indépendante de toute infrastructure d'exécution particulière, afin de permettre facilement son déploiement dans d'autres environnements, sans modifier la description d'application.

7.2.1 Éléments utiles pour décrire les applications GRIDCCM

Comme expliqué à la section 2.4.2.1 (page 31), une application GRIDCCM est une application CCM dont les composants peuvent avoir une ou plusieurs implémentations séquentielles ou parallèles. La description d'une application GRIDCCM comporte donc les éléments de la description des applications CCM (*cf.* section 2.2.2.5, page 16), ainsi que les informations propres à la technologie de programmation parallèle employée pour chaque implémentation parallèle de composant (PVM, MPI, OPENMP, *etc.*).

Pour chaque implémentation de composant parallèle, la description d'application GRIDCCM doit spécifier la technologie de programmation parallèle employée, ainsi que les gestionnaires de connexion. Comme l'illustre la figure 7.7, les gestionnaires de connexion *InputConnectionManager* et *OutputConnectionManager* de GRIDCCM permettent l'établissement des connexions entre les composants (séquentiels ou parallèles) de l'application[10]. Ces gestionnaires de connexion sont des composants au sens où nous l'avons précisé à la définition 2.3 (page 13) : leurs IDL et leurs implémentations doivent être spécifiés par la description spécifique d'application GRIDCCM.

[10]Ce sont des gestionnaires, et non des relais : pour éviter tout goulet d'étranglement, les processus des composants GRIDCCM s'échangent les données directement de l'un à l'autre, après avoir établi une connexion par l'intermédiaire d'un gestionnaire de connexion. Les gestionnaires de connexion servent à *établir les connexions* entre composants, et non à *échanger des données*. Se reporter au mémoire de thèse d'André Ribes [142].

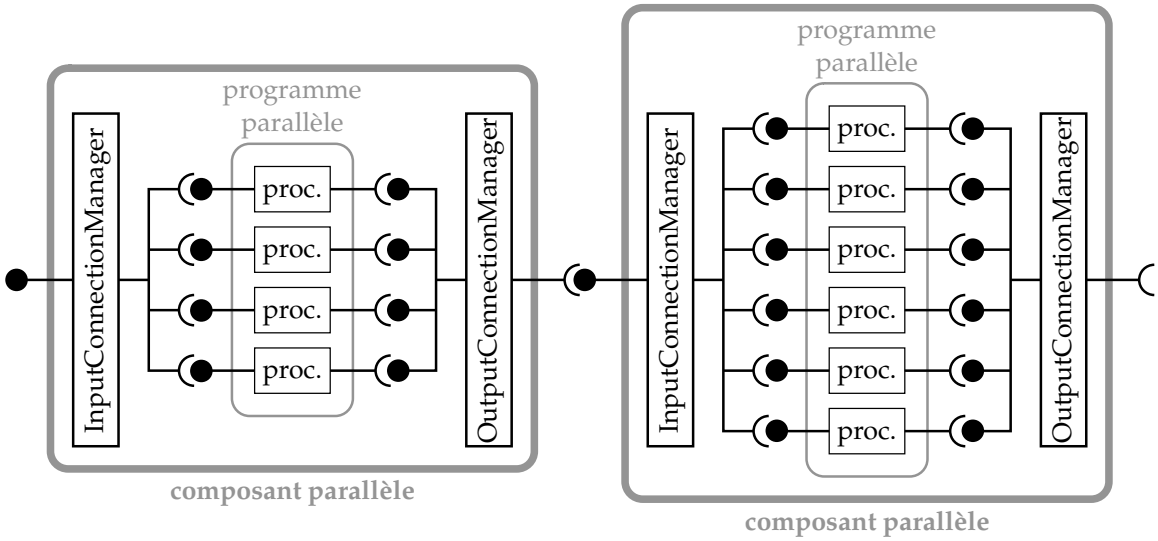


FIG. 7.7 – Deux composants GRIDCCM parallèles et leurs gestionnaires de connexion.

7.2.2 Modèle de description spécifique d'applications GRIDCCM

Comme une application GRIDCCM possède la *structure* d'une application CCM, nous choisissons de décrire une application GRIDCCM par le même formalisme que celui qui décrit les applications CCM (*cf.* section 2.2.2.5, page 16). La seule différence entre les deux descriptions porte sur la spécification des implémentations dans la description des composants (fichiers **`.csd`**). La description de l'implémentation séquentielle d'un composant reste la même que pour les applications CCM, avec l'élément **`<implementation>`** comme le montre la figure 2.3 (page 18). Une implémentation *parallèle* d'un composant voit son implémentation décrite par un élément **`<GridCCM_implementation>`** au lieu de **`<implementation>`**, comme l'illustre la figure 7.8.

L'élément `GridCCM_implementation` possède un identificateur `id` et un type `type`, qui précise s'il s'agit d'un composant parallélisé avec la technologie MPI, PVM, OPENMP, *etc.* Il contient aussi un élément `functional_prgrm` qui liste les localisations des codes métiers compilés du composant parallèle. L'élément `GridCCM_implementation` fait référence à une *technologie de parallélisation* plutôt qu'à une implémentation à proprement parler. En effet, l'élément `functional_prgrm` peut comporter lui-même plusieurs implémentations du composant, parallélisées en utilisant une même technologie (soit MPI, soit PVM, soit OPENMP, *etc.*). Enfin, les gestionnaires de connexion GRIDCCM sont listés : `input_connection_manager` et `output_connection_manager` peuvent eux aussi avoir plusieurs implémentations. Ces gestionnaires de connexion sont des composants identiques aux composants CCM habituels, donc ils sont décrits par des fichiers **`.csd`** dont les localisations sont précisées par les éléments `location`.

La figure 7.9 illustre un exemple de description d'implémentation parallèle d'un composant GRIDCCM. Cette « implémentation » parallèle du composant utilise la technologie MPI, et la description du programme MPI (telle que nous l'avons définie à la section 7.1) qui implémente ce composant parallèle se trouve à l'URL indiquée (HTTP). Les descriptions **`.csd`** des composants qui implémentent les gestionnaires de connexion peuvent être trouvées aux URL

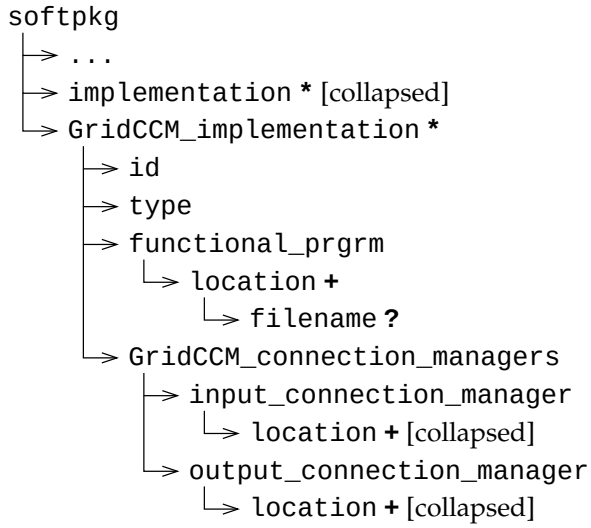


FIG. 7.8 – Hiérarchie du modèle de description d'implémentation parallèle de composant GRIDCCM.

spécifiées, par le protocole FTP.

7.2.3 Discussion

Ce modèle de description d'applications GRIDCCM peut être facilement étendu à des technologies de programmation parallèle autres que MPI : il suffit qu'un formalisme soit défini pour décrire les applications parallèles de cette technologie.

De même que les composants CCM traditionnels peuvent être implémentés sous la forme d'exécutables ou bien de DLL, les composants GRIDCCM peuvent exister sous les deux formes également. Par exemple, la description d'une application MPI spécifie le type des fichiers binaires de ses implémentations pour une utilisation éventuelle avec PadicoTM (*cf.* section 7.1.2.1).

Enfin, l'utilisation d'une implémentation parallèle de composant GRIDCCM est soumise à une restriction particulière par rapport aux composants CCM : une implémentation MPI de composant ne peut pas faire partie d'une « *process collocation* » de CCM (*cf.* section 2.2.2.5, page 16). En effet, une application MPI est constituée de *plusieurs processus distincts* qui ne partagent pas leurs espaces d'adressage virtuel. En revanche, un composant parallèle MPI peut faire partie d'une « *host collocation* ». Nous notons qu'une restriction du même ordre existe dans la norme CCM, puisque les composants CCM implémentés sous la forme d'exécutables (et non de DLL) ne peuvent pas appartenir à une « *process collocation* ».

7.3 Conclusion

Ce chapitre a présenté un format pour décrire les applications MPI et GRIDCCM de manière indépendante de toute infrastructure d'exécution. Ces informations sont nécessaires et suffisantes pour que la planification et le déploiement soient automatisables. Les descriptions spécifiques d'applications mêlent des informations utiles à la planification avec d'autres qui

```
<softpkg ...>
   ...
   <!-- Une implémentation séquentielle de ce composant GridCCM -->
   <implementation ...>
      ...
   </implementation>
   <!-- Une implémentation parallèle de ce composant GridCCM -->
   <GridCCM_implementation type="MPI" id="parallel_implem">
      <functional_prgrm>
         <location>http://parallel.components.org/FFT.mpi</location>
      </functional_prgrm>
      <GridCCM_connection_managers>
         <input_connection_manager>
            <location>ftp://ftp.cm.org/input-fft.csd</location>
         </input_connection_manager>
         <output_connection_manager>
            <location>ftp://ftp.cm.org/output-fft.csd</location>
         </output_connection_manager>
      </GridCCM_connection_managers>
   </GridCCM_implementation>
</softpkg>
```

FIG. 7.9 – Exemple de description de l'implémentation parallèle d'un composant GRIDCCM (extrait d'un fichier **.csd**).

ne sont utilisées qu'à l'exécution du plan de déploiement. La description spécifique d'une application ne fixe pas tout : elle peut laisser divers degrés de liberté, et le planificateur de déploiement sera responsable de faire des choix. De plus, les contraintes peuvent avoir différents degrés de nécessité, allant des contraintes qui doivent absolument être satisfaites aux contraintes indicatives qui ne sont pas impératives.

La spécification du formalisme de description des applications GRIDCCM est beaucoup plus courte que celle pour les applications MPI, car elle réutilise le format de description des applications CCM et celui des applications MPI. En effet, une application GRIDCCM n'est rien d'autre qu'une application CCM (de point de vue de sa structure), dont certains composants peuvent avoir une implémentation parallèle, utilisant la technologie MPI par exemple. Donc la description spécifique d'applications GRIDCCM est une simple *combinaison* des formalismes de description de CCM et MPI.

Les formalismes de description que nous avons présentés sont extensibles, notamment pour la spécification des *versions*. Par exemple, il peut être utile de préciser les versions pour :

- les fichiers de données en entrée et les fichiers binaires exécutables,
- les dépendances vis-à-vis de bibliothèques,
- les systèmes d'exploitation et les architectures matérielles pour lesquels les fichiers binaires sont compilés.

La spécification des versions doit pouvoir exprimer des contraintes telles que « la bibliothèque mathématique `libm` doit être au moins de version 2.3.1, sauf la version 2.3.4 », ou bien encore « le fichier binaire `fft.exe` est compatible avec toutes les architectures x86 sous Linux, mais il est optimisé pour les processeurs Pentium avec le support MMX[11] ». En général, plus les programmes compilés sont optimisés, et plus la spécification des versions doit être précise et avoir une grande expressivité. Par exemple, des relations d'ordre partiel peuvent être définies sur les versions pour exprimer les relations de compatibilité ascendante entre les versions de systèmes d'exploitation. Cette amélioration de notre formalisme de description spécifique d'applications pourrait s'inspirer de la description des paquets Debian[12].

Enfin, les contributions de ce chapitre ne sont pas restreintes aux grilles de calcul : les formalismes de descriptions spécifiques d'applications MPI et GRIDCCM sont valables pour tous types d'infrastructures d'exécution, puisqu'ils en sont indépendants. Simplement, les grilles de calcul ont été la motivation à définir ces formats de description, puisque ces infrastructures d'exécution peuvent accueillir des types d'applications très variés. La multitude des types d'applications à déployer sur les grilles de calcul nous a conduit à définir un formalisme de description *générique* d'application, qui sert d'unique interface au planificateur de déploiement, comme le montre le chapitre suivant.

[11] MMX : ***M**ulti**m**edia **E**xtensions*, un ensemble d'instructions machine de certains processeurs Intel, qui sont optimisées pour les calculs graphiques, le traitement du son et de la voix, *etc.*

[12] Debian est une distribution du système d'exploitation Linux et d'un grand nombre de logiciels (et non d'applications à déployer), tous packagés et décrits suivant un formalisme normalisé.

Chapitre 8

Description générique d'applications et plan de déploiement

Sommaire

Le chapitre précédent a présenté la notion de description *spécifique* d'application, en détaillant un formalisme pour les applications MPI et un autre pour les applications GRIDCCM : ces descriptions spécifiques contiennent toutes les informations utiles sur les applications pour leur déploiement automatique. Cependant, comme le chapitre 5 l'a souligné, un planificateur de déploiement est complexe à concevoir, donc nous voulons minimiser le nombre d'implémentations de planificateurs. Ainsi, nous proposons de convertir les descriptions spécifiques d'applications en une description générique, qui sert d'unique interface au planificateur, qui peut alors planifier le déploiement d'applications de différents types.

Ce chapitre présente notre modèle de description générique d'applications, ainsi que les convertisseurs qui permettent de traduire une description spécifique en une description générique. Ce modèle de description générique a un impact sur le plan de déploiement et sur le planificateur. Ce chapitre détaille donc ensuite notre formalisme de description du plan de déploiement, puis il revient sur le planificateur.

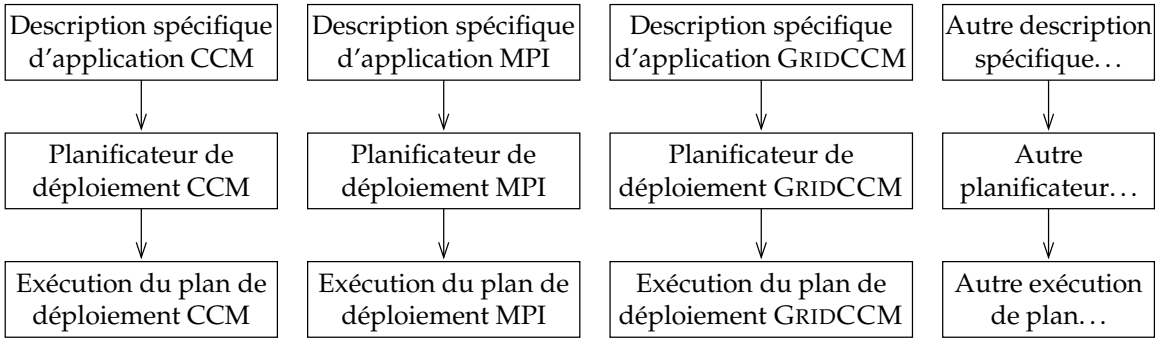


FIG. 8.1 – Autant de planificateurs spécifiques de déploiement que de types d'applications.

8.1 Description générique d'applications

Les sections précédentes ont présenté deux formats de description *spécifiques* adaptés aux applications parallèles MPI et aux applications mixtes GRIDCCM. Cette section explique l'intérêt de la description *générique* d'applications, puis elle définit son format. Ensuite, nous montrons comment les descriptions spécifiques d'applications peuvent être converties en une description générique pour les applications CCM, MPI, et GRIDCCM. Enfin, nous relatons une expérience réussie avec des applications JXTA de type pair-à-pair, qui met en évidence la généricité de notre modèle de description générique d'applications.

8.1.1 Motivations

8.1.1.1 Complexité et multitude des planificateurs de déploiement

L'intelligence de l'outil de déploiement réside dans la phase de planification. Comme nous l'avons expliqué à la section 5.3 (page 87), le planificateur de déploiement doit tenir compte des besoins exprimés par les applications, il doit prendre en compte les contraintes imposées par les ressources disponibles, et il doit satisfaire les exigences que l'utilisateur exprime au travers des paramètres de contrôle.

La section 5.3.2.2 (page 89) a souligné qu'il existe plusieurs *algorithmes* de planification. Par exemple, un algorithme de planification peut être déterministe ou non, il peut soit trouver une solution optimale ou simplement satisfaisante, il peut prendre en compte des paramètres de contrôle plus ou moins élaborés, *etc.* Pour un algorithme de planification donné, la figure 8.1 illustre qu'il faudrait *a priori* autant de planificateurs de déploiement qu'il existe de types d'applications. Or un planificateur est complexe à concevoir, du point de vue algorithmique, et à implémenter. Nous voulons donc *minimiser le nombre d'implémentations de planificateurs* de déploiement. Comme l'illustre la figure 8.2, la description *générique* d'applications vise à permettre de planifier le déploiement grâce à un seul et unique planificateur (pour un algorithme donné), valable pour n'importe quel type d'application. Il suffirait alors d'écrire, pour chaque type d'application, un simple convertisseur de description spécifique vers le format générique de description d'applications.

À la section 5.3.2.2 (page 89), nous avons introduit ce mécanisme sous le nom d'*isolation* du planificateur vis-à-vis des descriptions spécifiques d'applications. En plus de l'objectif d'isola-

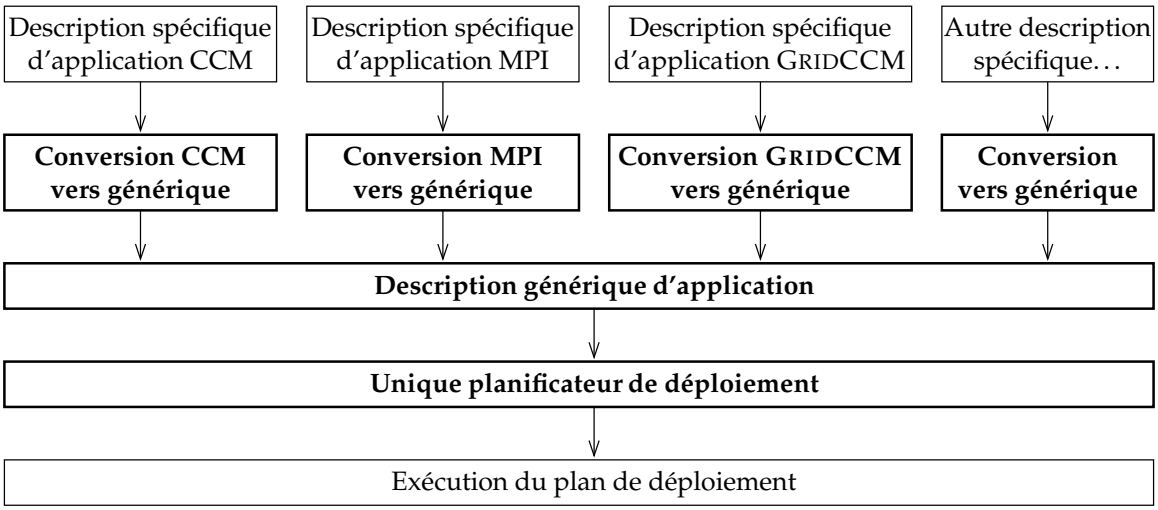


FIG. 8.2 – Notre modèle avec un unique planificateur générique de déploiement, et les convertisseurs de description spécifique vers générique d'applications.

tion du planificateur de déploiement vis-à-vis des différents types d'applications, le concept de description générique vise à *factoriser* les opérations de planification. En effet, sur la figure 8.1, les différents algorithmes de planification effectuent des opérations communes, telles que la vérification de la compatibilité entre les ressources et les versions compilées des programmes, ou encore la détermination de la cardinalité des instances de programmes. Ces opérations communes aux algorithmes de planification pourraient être factorisées dans une unique interface d'accès à la description générique d'applications.

En résumé, pour un algorithme de planification donné, notre modèle permet d'implémenter un unique planificateur de déploiement pour n'importe quel type d'application. En revanche, notre modèle n'interdit pas d'implémenter plusieurs planificateurs avec des *algorithmes de planification distincts*.

8.1.1.2 Applications mixtes

En ce qui concerne les applications mixtes telles que GRIDCCM, qui impliquent plusieurs technologies de programmation, nous ne voulons pas opter pour un planificateur de déploiement particulier d'applications CCM ou d'applications MPI, et nous ne voulons pas choisir une méthode de lancement propre aux applications distribuées ou bien aux applications parallèles. Le concept de description générique d'application donne lieu à un *modèle unifié de déploiement*, quel que soit le type d'application. Cette unification s'étend au-delà de la planification, comme le montre le chapitre 9, puisqu'elle concerne aussi l'exécution du plan de déploiement.

De plus, les applications mixtes peuvent être constituées d'un nombre non borné de technologies de programmation (PVM, OPENMP, *etc.*), ce qui nécessiterait tout autant de planificateurs de déploiement. Comme les planificateurs sont complexes à implémenter, il n'est pas souhaitable de multiplier les implémentations de planificateurs. Ainsi se renforce la nécessité

d'une interface unique pour le planificateur de déploiement : c'est la description générique d'applications.

8.1.2 Modèle de description générique d'applications

Cette section présente GADe (***G**eneric **A**pplication **De**scription*), notre modèle de description générique d'application. Ce modèle se calque sur le modèle d'exécution des applications dans les systèmes d'exploitation modernes : il se fonde sur le principe que toutes les applications se résument à un ensemble de processus, quels que soient leurs types. Ces processus peuvent charger des DLL dynamiquement, ou bien être contraints à s'exécuter sur un même nœud pour partager un banc mémoire, *etc.* Notre modèle de description générique d'application est une description de bas niveau, proche des concepts traditionnels des systèmes d'exploitation et des matériels, mais il n'est pas destiné à être exploité par l'utilisateur : c'est un document interne de l'outil de déploiement. GADe est débarrassé de tous les concepts spécifiques aux applications. Par exemple, il ne fait référence à aucun « composant », ou « port ».

8.1.2.1 Vue d'ensemble

Dans la description générique d'application, nous distinguons quatre entités : les processus, les éventuels codes à charger dans ces processus (DLL), les groupes de processus, et les interconnexions entre les groupes de processus.

Un **groupe de processus** est un ensemble de processus qui doivent s'exécuter sur la même machine, par exemple pour partager le même banc de mémoire physique. Cette notion de groupe de processus permet d'exprimer les contraintes de « *host collocation* » des applications CCM (*cf.* section 2.2.2.5, page 16).

Comme dans le monde des systèmes d'exploitation, un **processus** est une instance de programme en cours d'exécution. Un processus peut charger dynamiquement une ou plusieurs bibliothèques (ou DLL, ou code JAVA), que nous appelons « **codes à charger** »[1], et qui partagent le même espace d'adressage virtuel. Cette notion de processus qui peut renfermer plusieurs codes à charger permet par exemple d'exprimer les contraintes de « *process collocation* » des applications CCM.

En plus de lister les groupes de processus, les processus et les codes à charger, la description générique d'application spécifie les **connexions entre les groupes de processus**. Un groupe de processus est connecté à un autre groupe dès que l'un des processus d'un groupe est susceptible de communiquer avec au moins l'un des processus de l'autre groupe. Les connexions sont définies entre les groupes de processus et non entre les processus eux-mêmes. Même s'il est vrai, *in fine*, que ce sont les processus (et non les groupes de processus) qui établissent des connexions pour communiquer, il s'agit d'une question de configuration de l'application, dont le planificateur n'a pas besoin d'avoir connaissance. Il peut se contenter de savoir quelles machines doivent pouvoir communiquer entre elles. Ainsi, nous considérons que si un processus d'un groupe sur une machine peut communiquer avec un autre processus d'un autre groupe sur une machine distincte, alors tous les processus des deux groupes

[1] Cette appellation de « codes à charger » permet de s'affranchir des environnements d'exécution : systèmes d'exploitation Windows ou UNIX, ou bien machine virtuelle JAVA (JVM).

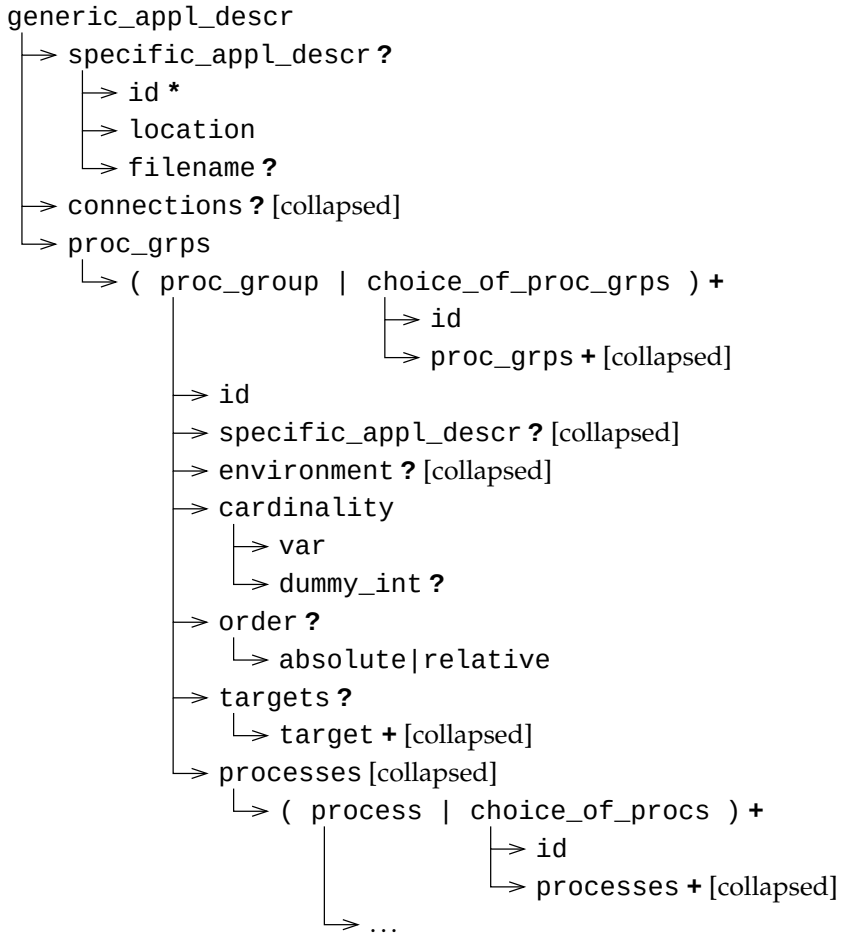


FIG. 8.3 – Hiérarchie du modèle de description générique d'application : les groupes de processus.

peuvent communiquer entre eux. De même, les connexions ne se définissent pas entre les codes à charger.

8.1.2.2 Spécification du modèle de description générique d'application

La figure 8.3 montre une partie de la hiérarchie de GADe. Une description générique est issue d'une description spécifique d'application : l'élément `specific_appl_descr` permet de retrouver la description spécifique d'application originale (avec ses éléments `location` et `filename`, qui permet de désigner un fichier à l'intérieur d'une archive ZIP). C'est utile lors de la phase de configuration du déploiement : il est nécessaire de revenir à la description spécifique d'application pour la configuration, car cette phase est spécifique au type d'application (*cf.* section 9.4, page 172).

Groupes de processus. L'élément `proc_grps` liste les différents groupes de processus de la description générique. Les groupes de processus `proc_group` nécessairement présents sont des fils directs de `proc_grps`. Mais il se peut que le planificateur de déploiement ait le choix

de déployer un groupe de processus parmi plusieurs : c'est le rôle de l'élément `choice_of_proc_grps`, qui laisse le choix au planificateur entre plusieurs groupes de processus. Cette fonctionnalité permet par exemple de décrire des composants GRIDCCM qui possèdent à la fois une implémentation séquentielle (groupe de cardinalité 1) et une implémentation parallèle (groupe de cardinalité supérieure à 1) : les implémentations sont décrites dans le paragraphe suivant, au sein des processus. Le planificateur est responsable de choisir l'un ou l'autre groupe de processus.

Un groupe de processus possède une cardinalité C_{grp}, qui indique combien de fois il doit être répliqué. L'expression de la cardinalité C_{grp} peut adopter le même formalisme que celui employé pour les applications MPI. Les C_{grp} groupes de processus peuvent être déployés sur des machines distinctes, à la discrétion du planificateur. Il est important de noter que les C_{grp} instances du groupe de processus ne sont pas forcément identiques : si le planificateur a le choix des processus et de leurs implémentations au sein d'un groupe de processus, il se peut qu'une partie des C_{grp} groupes soit déployée sur des IBM sous AIX et que l'autre partie le soit sur des PC sous Linux, et donc avec d'autres versions compilées des programmes (ou implémentations).

L'élément `id` permet de désigner les groupes de processus, par exemple pour la description des connexions entre les groupes. L'élément `proc_group` possède aussi un élément `specific_appl_descr`, qui peut désigner, si besoin, l'objet auquel il correspond dans la description spécifique d'application. L'élément `environment` est le même que pour les applications MPI (*cf.* section 7.1.2.1) : son rôle dépasse la planification de déploiement, il permet surtout de réaliser une partie de l'exécution du plan.

Dans le cas où il existe des dépendances (flux de données par exemple) entre les programmes d'une application à déployer, l'élément `order` permet de définir l'ordre dans lequel les groupes de processus doivent être lancés. C'est utile par exemple avec les applications CORBA qui requièrent un service de nommage (*cf.* section 4.2.1.3, page 56) : ce service est lancé en premier, et sa référence (IOR) est donnée aux autres programmes CORBA lancés ensuite. L'ordre peut être une valeur absolue (les groupes d'ordre 10 doivent être déployés avant les groupes d'ordre 43), ou bien une valeur relative, indiquant que le groupe de processus en cours de description doit être déployé avant ou après un autre groupe dont l'identificateur est donné.

L'élément `targets` est identique à celui de la description spécifique des applications MPI (*cf.* section 7.1.2.1). Cet élément n'est pas strictement nécessaire à ce niveau de la description (*i.e.* dans les groupes de processus), puisque les processus qui appartiennent au groupe ont des implémentations qui spécifient pour quelles machines elles sont compilées. Si un processus ne possède pas de code à charger, alors l'ensemble des machines cibles (« *target* » T_{proc}) de ce processus est l'union des cibles des implémentations possibles du processus :

$$T_{proc} = \bigcup_{proc\_impl \in proc} T_{proc\_impl}.$$

S'il possède des codes à charger, qui ont eux aussi une implémentation, alors l'ensemble des cibles du processus est :

$$T_{proc} = \left(\bigcup_{proc\_impl \in proc} T_{proc\_impl} \right) \bigcap \left(\bigcup_{code \in proc} T_{code\_impl} \right).$$

Ainsi, l'ensemble des machines cibles listées au niveau d'un groupe de processus est l'intersection des cibles des processus du groupe, puisque tous ces processus doivent s'exécuter sur la même machine : $T_{grp} = \bigcap_{proc \in grp} T_{proc}$. La présence de l'élément `targets` au niveau des groupes de processus permet de faciliter la tâche du planificateur, qui peut tester rapidement si un groupe peut être placé sur une ressource de système d'exploitation et d'architecture matérielle donnée.

Enfin, l'élément `processes` énumère les différents processus qui appartiennent au groupe. Les processus `process` fils directs de `processes` doivent tous faire partie du groupe de processus. Mais il se peut que le planificateur soit chargé de choisir un processus parmi plusieurs pour faire partie du groupe : c'est le rôle de l'élément `choice_of_procs`, qui liste des ensembles de processus, dont un seul sera sélectionné par le planificateur de déploiement.

Processus. Comme l'illustre la figure 8.4, l'élément `process` possède, de même que les groupes de processus, un identificateur `id`, une cardinalité `cardinality` (nombre de fois que le processus doit être répliqué dans le groupe), un environnement, un ordre de lancement au sein du groupe, et un élément `specific_appl_descr`, qui peut désigner, si besoin, l'objet auquel le processus correspond dans la description spécifique d'application.

Un processus peut avoir plusieurs versions compilées pour des cibles différentes, listées dans `implementations`. Chaque implémentation possède un identificateur, un environnement, et un pointeur vers l'objet qui lui correspond dans la description spécifique d'application. Pour chaque implémentation, l'élément `binary_location` spécifie la ou les localisations du fichier (binaire) de l'implémentation. Par exemple, un exécutable peut être récupéré sur un site par le protocole HTTP, ou sur un autre site par le protocole FTP. Enfin, l'implémentation d'un processus indique la liste des machines cibles `targets` sur lesquelles elle peut s'exécuter. Si aucune cible n'est précisée, alors l'implémentation est compatible avec n'importe quelle machine (par exemple un script Python, ou bien du bytecode JAVA si l'environnement de l'implémentation spécifie une dépendance sur une JVM). La structure de l'élément `target` est identique à celui de la description spécifique des applications MPI (*cf.* section 7.1.2.1).

Codes à charger. La figure 8.4 montre qu'un processus peut comprendre un ou plusieurs codes à charger, qui sont listés par l'élément `codes_to_load`. L'élément `choice_of_codes_to_load` permet de laisser au planificateur la responsabilité de sélectionner un ensemble de codes à charger parmi plusieurs. Chaque code `code_to_load` peut spécifier l'objet de la description spécifique d'application auquel il correspond par le biais de l'élément `specific_appl_descr`. Comme pour les processus, un code à charger possède un identificateur `id`, une cardinalité `cardinality` et une liste d'implémentations, dont une devra être choisie par le planificateur de déploiement.

Connexions. Comme l'illustre la figure 8.5, la rubrique `connections` liste les connexions entre les groupes de processus. Chaque connexion s'applique à plusieurs groupes de processus : les éléments `ref_id` servent à indiquer quels groupes sont concernés par la connexion en cours de description. Pour chaque groupe référencé, l'élément `role` peut indiquer si ce processus est susceptible d'initier la connexion réseau (rôle de « client »), d'attendre les connexions entrantes (rôle de « serveur »), ou bien les deux. Cette information est utile au planificateur

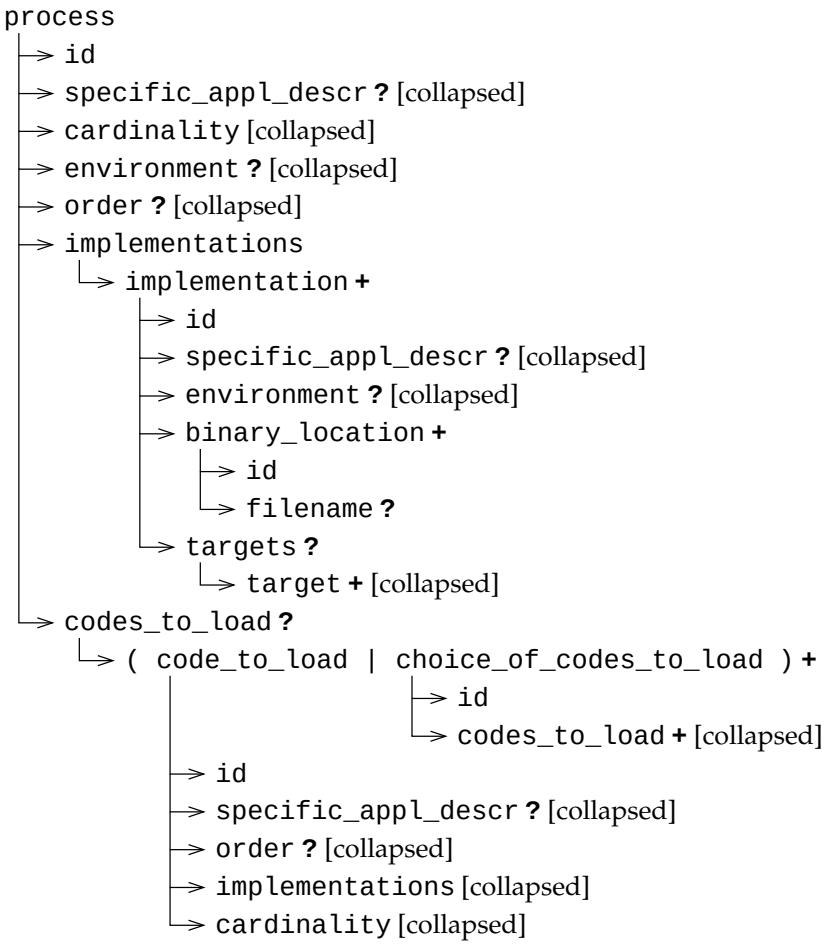


FIG. 8.4 – Hiérarchie du modèle de description générique d'application : les processus.

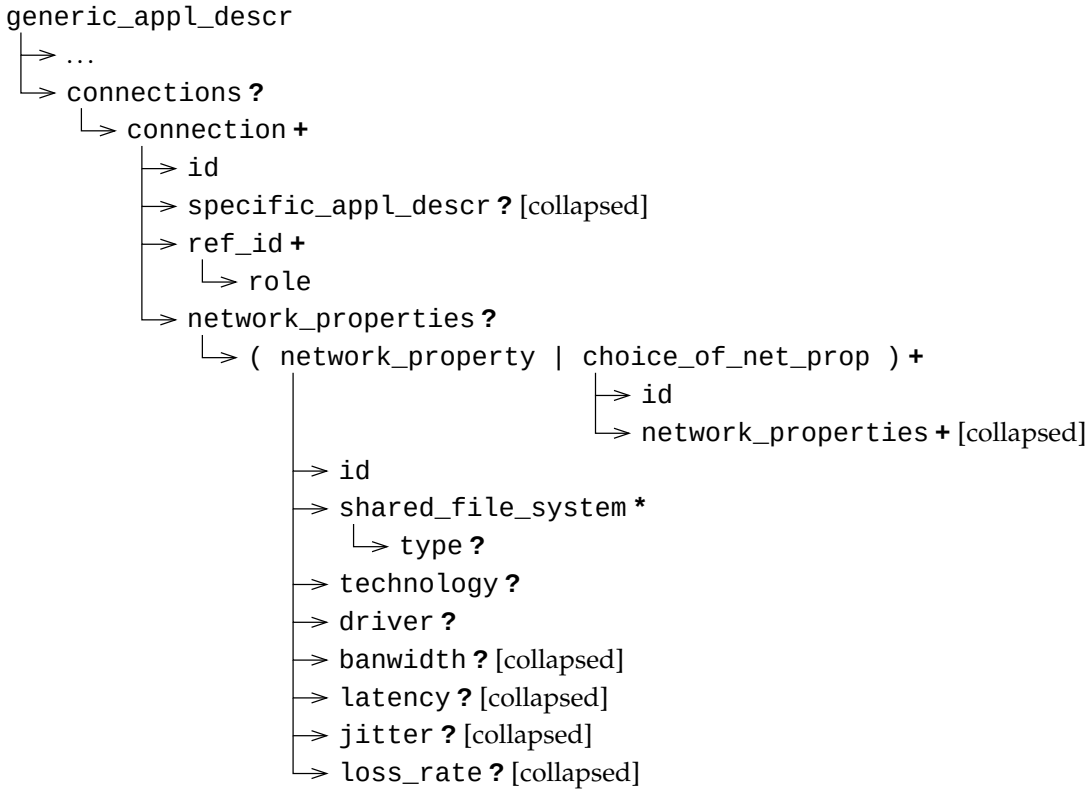


FIG. 8.5 – Hiérarchie du modèle de description générique d'application : les connexions.

pour savoir quels groupes de processus peuvent être placés derrière des pare-feux (*cf.* figure 8.6).

Enfin, la liste des propriétés réseau requises est spécifiée par `network_properties`. Le planificateur peut avoir le choix entre plusieurs modes de connexion (avec `choice_of_net_prop`) : les éléments `network_property` peuvent autoriser différentes technologies réseau entre les groupes de processus. L'élément `network_property` permet également de préciser que tous les processus doivent avoir accès au même réseau Myrinet, ou bien qu'ils doivent être déployés avec une technologie MPI particulière, ou encore qu'ils ont besoin de tel débit ou telle latence (valeur minimale, valeur maximale, moyenne, variance dans le temps, *etc.*).

```
<connections>
  <connection id="client-server">
    <ref_id role="source">client1_proc_group</ref_id>
    <ref_id role="source">client2_proc_group</ref_id>
    <ref_id role="destination">server_proc_group</ref_id>
  </connection>
</connections>
```

FIG. 8.6 – Les groupes de processus clients initient les communications, et le groupe de processus serveur les reçoit.

8.1.3 Conversion d'une description spécifique vers une description générique

Les sections précédentes ont donné une vue d'ensemble et la spécification de GADe, notre modèle de description spécifique d'application. Des convertisseurs, spécifiques à chaque type d'application, sont responsables de traduire des descriptions spécifiques d'application en descriptions génériques. Cette section montre, pour différents types d'applications (MPI, CCM, GRIDCCM), comment la description spécifique d'application peut être traduite en description générique.

8.1.3.1 Applications parallèles MPI

Comme le format de description générique est proche de celui de description spécifique d'application MPI, la conversion d'un format à l'autre est simple.

Comme l'illustre la figure 8.7, une application MPI constituée de n processus (dans la description spécifique) se traduit en un groupe de processus : la cardinalité du groupe est n, et ce groupe comprend un seul processus, de cardinalité 1. En effet, pour les applications MPI, il n'est en général pas nécessaire que les processus partagent la même machine, donc il n'est pas utile de rassembler les processus dans le même groupe. Le processus de la description générique ne possède pas de code à charger si le programme MPI consiste en un exécutable. En revanche, le processus peut comporter plusieurs implémentations, compatibles avec différents systèmes d'exploitation et architectures matérielles. Le fichier binaire désigné dans cet exemple est lié dynamiquement à la bibliothèque MPICH-GM, qui constitue une dépendance.

La description générique comporte un élément `connection` qui référence l'unique groupe de processus. Cette déclaration signifie que toutes les n instances du groupe devront pouvoir communiquer entre elles. L'élément `role` indique que chaque instance de groupe est susceptible d'initier les connexions réseau. Pour une application MPICH-GM, destinée à s'exécuter sur un cluster Myrinet, les propriétés réseau indiquent que les n instances du groupe de processus doivent être placées au sein d'un unique cluster Myrinet dont les nœuds possèdent le pilote GM.

8.1.3.2 Applications distribuées CCM

La figure 8.8 illustre un exemple d'application CCM par une représentation schématique de sa description spécifique. Cette application est constituée d'une *host-collocation* A, qui contient un composant 1 sous la forme de DLL, un composant 2 sous la forme d'exécutable, ainsi qu'une *process-collocation* B comprenant deux composants sous la forme de DLL (3 et 4). L'application est également constituée d'un composant 7 exécutable, et d'une *process-collocation* C comprenant deux composants sous la forme de DLL (5 et 6). Les paires de composants 3 et 5, 2 et 6, 6 et 7 ont leurs ports interconnectés.

La figure 8.9 donne une représentation schématique de la description générique de cette application CCM. Une *host-collocation* se traduit en un groupe de processus (A), et une *process collocation* se convertit en un processus (B). Les composants sous la forme de DLL (ou classe JAVA) donnent des codes à charger : ils appartiennent à un processus dont l'implémentation est celle d'un serveur de conteneur (*ComponentServer*, *cf.* section 2.2.2.5, page 16) : c'est le cas par exemple des codes à charger 5 et 6, qui sont insérés dans le processus C dont l'implémentation

```
<generic_appl_descr>
  <proc_grps>
    <proc_group id="mpi_group">
      <cardinality var="n" dummy_int="i"> n = 2*i & i > 1 </cardinality>
      <processes>
        <process id="mpi_proc">
          <cardinality var="n"> n = 1 </cardinality>
          <implementations>
            <implementation id="mpi_impl">
              <binary_location>
                <location>http://mpi.store.org/linux-fft.exe</location>
              </binary_location>
              <environment>
                <dependency>mpich-gm-1.2.6</dependency>
              </environment>
              <targets>
                <target>
                  <operating_system>
                    <OS_name>Linux</OS_name> <OS_bits>32</OS_bits>
                  </operating_system>
                  <computer_ISA>
                    <ISA_name>x86</ISA_name>
                  </computer_ISA>
                </target>
              </targets>
            </implementation>
          </implementations>
        </process>
      </processes>
    </proc_group>
  </proc_grps>
  <connections>
    <connection id="mpi_connect">
      <ref_id role="both">mpi_group</ref_id>
      <network_properties>
        <network_property id="gm_net_prop">
          <driver>myrinet-gm</driver>
        </network_property>
      </network_properties>
    </connection>
  </connections>
</generic_appl_descr>
```

FIG. 8.7 – Exemple de description générique d'application au format XML, correspondant à une application parallèle MPICH-GM.

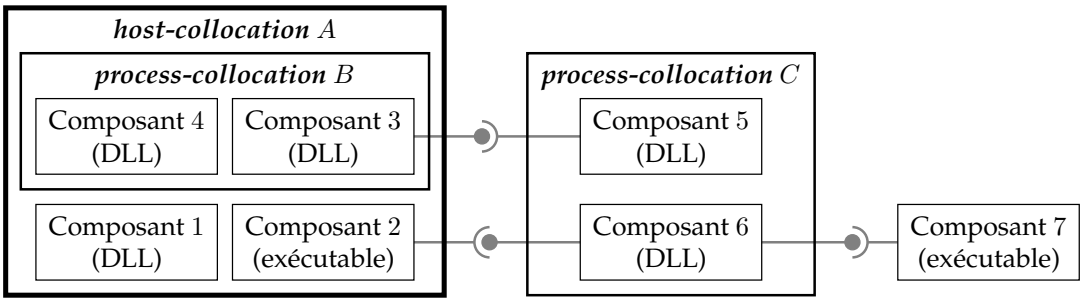


FIG. 8.8 – Exemple schématique d'application distribuée CCM, décrite dans le formalisme spécifique des applications CCM.

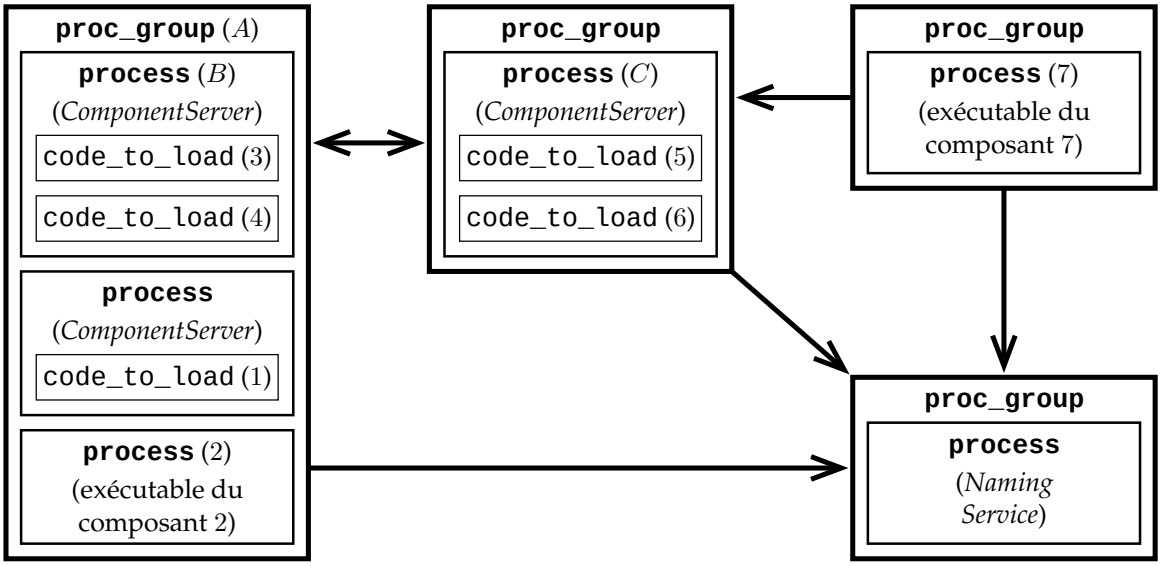


FIG. 8.9 – Représentation schématique de la description générique de l'application distribuée CCM de la figure 8.8.

est un serveur de conteneur. Les composants sous la forme d'exécutables sont traduits en processus sans code à charger. Comme le composant 7 n'a aucune contrainte de co-localisation par rapport aux autres composants, il est inséré dans un groupe de processus indépendant qui ne comprend qu'un seul processus.

La description spécifique d'une application CCM ne décrit pas explicitement les serveurs de conteneur : conformément à la norme CCM, il est implicite que les composants sous la forme de DLL s'exécutent au sein d'un serveur de conteneur. Il revient donc à l'utilisateur de préciser, par le biais des paramètres de contrôle, où l'outil de déploiement peut trouver un tel serveur de conteneur. En pratique, un serveur de conteneur peut être packagé et décrit comme un composant exécutable dans un fichier `.csd`. L'utilisateur peut donc se contenter d'indiquer, dans les paramètres de contrôle, la localisation du package qui contient le serveur de conteneur et sa description spécifique.

Il en est de même pour le service de nommage (*Naming Service*, *cf.* section 4.2.1.3, page 56) :

```
<connections>
  <!-- Connexions entre les paires de composants 3 et 5, 2 et 6 -->
  <connection id="A_C">
    <ref_id role="both">proc_group_A</ref_id>
    <ref_id role="both">proc_group_with_process_C</ref_id>
  </connection>
  <!-- Connexion entre les composants 6 et 7 -->
  <connection id="7_to_C">
    <ref_id role="source">prog_group_with_process_7</ref_id>
    <ref_id role="destination">proc_group_with_process_C</ref_id>
  </connection>
  <!-- Connexions depuis tous les composants
                  vers le service de nommage -->
  <connection id="all_to_NS">
    <ref_id role="source">proc_group_A</ref_id>
    <ref_id role="source">proc_group_with_process_C</ref_id>
    <ref_id role="source">prog_group_with_process_7</ref_id>
    <ref_id role="destination">proc_group_of_naming_service</ref_id>
  </connection>
</connections>
```

FIG. 8.10 – Connexions de l'application CCM de la figure 8.8 exprimées au format XML.

il peut être packagé et décrit comme un composant exécutable dans un fichier `.csd`. L'utilisateur peut alors préciser la localisation du package du service de nommage dans les paramètres de contrôle. Ainsi, le convertisseur de description spécifique d'application CCM peut ajouter le service de nommage parmi les éléments à déployer. Le convertisseur doit aussi indiquer un ordre de déploiement : le groupe de processus du service de nommage doit être lancé *avant* tous les autres groupes, car sa référence (IOR) devra être transmise aux composants au moment de leur lancement.

Enfin, les connexions entre les groupes de processus reflètent les interconnexions de ports entre les composants (*cf.* figure 8.10). En particulier, le groupe de processus qui contient le service de nommage doit pouvoir communiquer avec chacun des composants, donc avec chacun des groupes de processus de la description générique.

8.1.3.3 Applications mixtes GRIDCCM

Pour illustrer le principe du convertisseur de description spécifique d'application GRIDCCM en description générique, nous reprenons l'exemple de la figure 7.9. Il s'agit d'un composant GRIDCCM qui possède une implémentation séquentielle, et une implémentation parallèle MPI.

La figure 8.11 montre un extrait de la description générique de cette application GRIDCCM. Parmi les groupes de processus se trouve un élément `choice_of_proc_grps` qui liste deux ensembles de groupes de processus. Le premier ensemble est constitué d'un seul groupe de cardinalité 1 : il représente l'implémentation séquentielle du composant. Le second ensemble est constitué d'un groupe de processus de cardinalité multiple : il s'agit de l'implémentation parallèle du composant GRIDCCM. Le planificateur de déploiement sera res-

```
<generic_appl_descr>
  <proc_grps>
    <choice_of_proc_grps id="the_gridccm_component">
      <!-- Choisir soit cette version séquentielle du composant... -->
      <proc_grps>
        <proc_group id="sequential_group">
          <cardinality var="n"> n = 1 </cardinality>
          ...
        </proc_group>
      </proc_grps>
      <!-- ... soit cette version parallèle -->
      <proc_grps>
        <proc_group id="parallel_group">
          <cardinality var="n"> n >= 8 & n <= 256 </cardinality>
          ...
        </proc_group>
      </proc_grps>
    </choice_of_proc_grps>
    <!-- Un autre composant... -->
    <proc_group id="another_component">
      ...
    </proc_group>
  </proc_grps>
  <connections>
    <connection id="bidirect_connection">
      <ref_id role="both">parallel_group</ref_id>
    </connection>
    ...
  </connections>
</generic_appl_descr>
```

FIG. 8.11 – Exemple de description générique d'application au format XML, correspondant à une application GRIDCCM.

ponsable de choisir l'un ou l'autre de ces deux ensembles. Enfin, la rubrique des connexions indique que toutes les instances du groupe de processus `parallel_group` doivent pouvoir communiquer, si ce groupe est sélectionné.

8.1.4 Discussion

8.1.4.1 Retrouver les objets de la description spécifique d'application

Comme nous venons de le voir, la description générique d'application ne contient pas toutes les informations de la description spécifique. En effet, elle ne sert qu'à la planification et à une partie de l'exécution du plan de déploiement. Ainsi, pour la phase de configuration, il est nécessaire de faire le lien entre les objets de GADe (connexions, codes à charger, processus et

groupes de processus) et les objets des descriptions spécifiques, afin de pouvoir remonter des processus lancés vers les composants d'une application CCM par exemple. Cette correspondance est possible grâce aux éléments `specific_appl_descr` que le convertisseur prend soin d'associer à chaque objet de GADe.

8.1.4.2 Tout rendre explicite

Dans la description spécifique d'une application MPI, il n'est pas utile de spécifier que tous les processus doivent être en mesure de communiquer entre eux, car c'est implicite d'après la norme MPI : le convertisseur de description spécifique MPI rend cette information explicite par le biais des connexions. De même, la description des applications CCM ne mentionne ni le service de nommage, ni les serveurs de conteneur pour les composants sous la forme de DLL : le convertisseur spécifique des applications CCM rend toutes ces informations explicites.

Ainsi, les convertisseurs de chaque type d'application sont responsables de rendre explicites tous les objets à déployer et toutes les contraintes applicatives (dépendances et connexions). Par conséquent, le planificateur n'est pas *alourdi* par les concepts spécifiques des différents types d'applications, et donc plus facile et systématique à écrire.

8.1.4.3 Expérience réussie avec JXTA

Cette section relate une expérience réussie avec des applications JXTA, qui met en évidence la généricité de GADe. JXTA [201] est une bibliothèque de communication pair-à-pair (P2P).

Mathieu JAN[2] a défini un langage simple de description spécifique d'applications JXTA, qui s'exprime en termes de pairs de rendez-vous, de relais, *etc.* Ensuite, il a écrit un convertisseur de description spécifique JXTA en description générique d'applications (GADe) sous la forme de plugin dans notre outil de déploiement ADAGE (*cf.* chapitre 10). C'est donc à moindre frais[3] qu'un nouveau type d'application a été ajouté aux applications supportées par ADAGE, sans avoir besoin d'écrire de planificateur de déploiement spécifique aux applications JXTA.

GADe a été conçu avec les applications CCM, MPI, et GRIDCCM en ligne de mire. Cette expérience réussie démontre l'intérêt pratique de GADe, qui n'est pas limité aux types d'applications parallèles et/ou distribuées. Notre modèle de description générique remplit donc bien son rôle d'interface unique entre la description spécifique des applications et le planificateur de déploiement, au prix d'un simple convertisseur.

8.1.4.4 Limites de la généricité

Les applications XCAT3, Ccaffeine ou DCA de CCA (*cf.* sections 2.2.2.4 page 14, et 2.4.2.2 page 32) ne font pas intervenir de concepts que GADe ne sache pas exprimer. Il en est de même pour les applications ProActive en JAVA par exemple. Cependant, il serait intéressant d'étudier précisément si GADe est à même de supporter ces autres types d'applications.

[2]Doctorant à l'IRISA / INRIA (Rennes), dans le projet Paris.

[3]Le plugin JXTA compte moins de 800 lignes de code C++ en tout, dont une partie relativement verbeuse pour analyser du XML et y rechercher des données.

Plus généralement, nous estimons que GADe peut représenter un grand nombre de types d'applications *statiques*, puisque ce modèle de description n'est pas lié aux applications (GADe est indépendant de la nature parallèle, distribuée, pair-à-pair, *etc.* des applications), mais aux infrastructures d'exécution (matériels et systèmes d'exploitation). Sur les infrastructures que nous visons, toutes les applications statiques se résument à des processus et des codes à charger, avec des dépendances et contraintes de co-localisation et de communication éventuelles. GADe n'a pas été conçu pour des applications *dynamiques*.

Ainsi, nous situons la limite de GADe au niveau des infrastructures d'exécution : ce modèle de description générique d'application peut ne plus être valable hors du contexte des systèmes d'exploitation actuels et de la vision moderne des ordinateurs.

8.1.4.5 Amélioration du pouvoir d'expression

Nous avons conscience de deux limitations de l'état actuel de GADe. D'une part, le pouvoir d'expression de la cardinalité dans GADe ne permet pas encore d'exprimer toutes les contraintes que permet de spécifier la description spécifique des applications MPI. D'autre part, nous n'avons pas mené d'investigations approfondies sur la traduction de la description du partitionnement et des topologies virtuelles MPI en contraintes sur les propriétés réseau exprimées au sein des connexions de GADe.

En l'état actuel, GADe ne permet pas d'exprimer, par le biais des propriétés réseau des connexions, les contraintes de partitionnement et de topologies virtuelles MPI avec la même finesse et les mêmes degrés de liberté que la description spécifique d'applications MPI. Bien que ces contraintes ne soient pas impératives pour le fonctionnement correct d'une application MPI, elles sont intéressantes à exprimer dans un contexte de grille de calcul, où les ressources sont hétérogènes. Par exemple, la figure 7.6 spécifie, de manière indicative, le partitionnement souhaité pour une application MPI : les processus esclaves ne doivent pas être dispersés sur plus de quatre clusters ou sur plus de quatre sites. En l'état actuel de GADe, la description générique d'application n'exprime pas cette contrainte avec tous ses degrés de libertés, et le convertisseur de description spécifique d'application MPI doit faire des choix : il doit décider du partitionnement exact de l'application (processus répartis sur 1, 2, 3 ou 4 clusters ou sites), et il doit préciser les caractéristiques réseau souhaitées entre les partitions de processus MPI (partitionnement statique sur les clusters Myrinet d'un même réseau local, ou sur les sites d'un même pays).

En toute rigueur, le convertisseur ne devrait pas avoir à faire ces choix, et laisser la responsabilité de prendre ces décisions au planificateur de déploiement. L'enrichissement de GADe pour qu'il puisse exprimer ces contraintes et ces degrés de liberté est une perspective à étudier.

8.2 Retours sur la planification et le plan de déploiement

Maintenant que notre modèle de description générique d'application a été présenté, nous pouvons revenir sur le plan de déploiement et le planificateur. En effet, le plan et le planificateur de déploiement ont été partiellement introduits à la section 5.3 (page 87), mais le format du plan de déploiement est intimement lié à la description générique d'application, et le planificateur s'interface directement avec cette description générique.

Cette section présente le format du plan de déploiement, puis elle mentionne quelques opérations élémentaires que nous avons identifiées pour les planificateurs, afin de simplifier leurs implémentations.

8.2.1 Description du plan de déploiement

Le plan de déploiement est une liste d'associations entre les groupes de processus de la description générique d'application et les ressources d'exécution qui ont été sélectionnées par le planificateur. En outre, le plan de déploiement spécifie, pour chaque groupe de processus, sa cardinalité, l'implémentation sélectionnée pour chacun des processus et des codes à charger, ainsi que leurs cardinalités et les localisations retenues pour rapatrier les fichiers binaires. Pour chaque ressource sélectionnée, le plan de déploiement spécifie la méthode de soumission et le protocole de transfert de fichiers choisis par le planificateur.

La figure 8.12 illustre la structure hiérarchique du plan de déploiement. L'élément `generic_appl_descr` pointe vers la description générique d'application à laquelle correspond le plan de déploiement. En effet, un plan n'a de sens que par rapport à une description générique d'application, puisqu'il établit des associations entre les groupes de processus d'une description générique et des ressources de calcul.

L'élément `proc_grp_association` liste les différents associations d'un groupe de processus à diverses ressources de calcul. Un groupe de processus possède une cardinalité qui peut être supérieure à 1, et chaque instance de ce groupe peut être placée sur des ressources distinctes. Ainsi, cet élément `proc_grp_association` regroupe les différentes ressources sélectionnées pour le groupe de processus dont l'identificateur est indiqué par `ref_id`. L'élément `order` spécifie, par une valeur numérique entière, l'ordre dans lequel les groupes de processus doivent être déployés.

Pour un groupe de processus donné, un ou plusieurs éléments `association` associent un certain nombre d'instances du groupe de processus à une ressource d'exécution précise. Le nombre d'instances du groupe de processus associées à cette ressource est donné par l'élément `cardinality`.

8.2.1.1 Sélection des processus au sein d'un groupe de processus

L'élément `processes` liste les processus qui ont été sélectionnés par le planificateur pour faire partie de ces instances de groupe de processus. L'élément `ref_id` pointe vers un identificateur de processus de la description générique d'application ; `order` définit un ordre de lancement des processus au sein du groupe de processus, par le biais d'une valeur entière (soit fixée par la description générique d'application, soit fixée par le planificateur de déploiement) ; `cardinality` donne le nombre d'instances de ce processus à lancer dans ce groupe de processus. Une implémentation précise du processus est spécifiée par son identificateur avec `ref_id`, ainsi que la localisation choisie pour rapatrier le fichier binaire du processus.

Il en est de même pour les codes à charger : ils sont référencés par les identificateurs de la description générique d'application, un ordre et une cardinalité précis leur sont affectés, et une implémentation est sélectionnée.

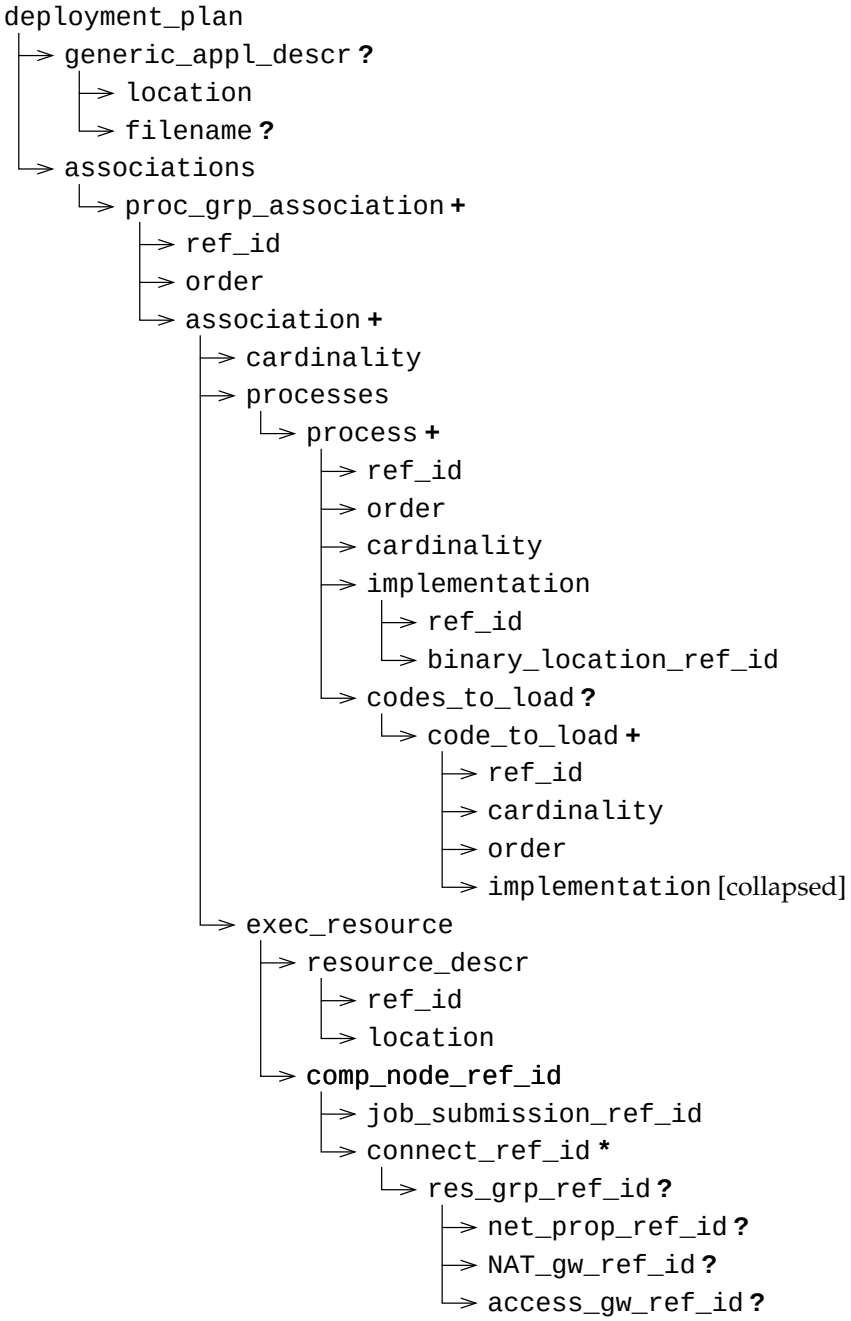


FIG. 8.12 – Structure hiérarchique du plan de déploiement.

8.2.1.2 Sélection d'une ressource d'exécution associée à des instances d'un groupe de processus

L'élément `exec_resource` spécifie la ressource d'exécution que le planificateur a sélectionnée pour y placer les instances du groupe de processus. L'élément `resource_descr` permet de référencer le descripteur dans lequel la ressource d'exécution est définie. L'élément `comp_node_ref_id` pointe vers le nœud de calcul sur lequel seront lancés les processus du groupe de processus, et `job_submission_ref_id` spécifie l'identificateur de la méthode de soumission de tâche à distance sélectionnée par la planificateur pour lancer les processus sur le nœud de calcul.

Les éléments `connect_ref_id` donnent les identificateurs des connexions de la description générique d'application qui ont été choisies par le planificateur. Il est utile de pouvoir attacher plusieurs connexions à un groupe de processus pour des applications qui mêlent à la fois des communications CORBA sur Ethernet, et des communications MPI sur Myrinet. Par exemple, les composants parallèles GRIDCCM peuvent avoir recours à deux telles technologies réseau. L'élément `res_grp_ref_id` référence le groupe réseau parent du nœud de calcul (`comp_node_ref_id`) qui spécifie l'interface réseau qui doit être utilisée pour communiquer avec le nœud de calcul. En effet, un nœud de calcul ne peut être relié à un groupe réseau que par une seule et unique interface réseau. Enfin, `NAT_gw_ref_id` et `access_gw_ref_id` identifient les passerelles de traduction d'adresses ou d'accès aux réseaux sous adresses IP privées que le planificateur a sélectionnées, si nécessaire.

8.2.1.3 Discussion

Nous retrouvons bien dans le formalisme de description du plan de déploiement la *spécification de tous les choix* que doit faire le planificateur :

- sélection des groupes de processus, des processus, et des codes à charger ;
- pour chacun de ces éléments, détermination des cardinalités et des ordres de lancement ;
- sélection des implémentations des processus et des codes à charger, ainsi que la localisation des fichiers binaires à rapatrier ;
- choix des machines d'exécution, avec leurs méthodes de soumission de tâches à distance et leurs interfaces réseau.

Si la description des ressources était étendue pour spécifier les méthodes de transfert de fichiers des nœuds de calcul (*cf.* section 6.3.7, page 117), le plan de déploiement devrait lui aussi être étendu pour spécifier, avec la méthode de soumission de tâches, la méthode sélectionnée pour les transferts de fichiers.

8.2.2 Opérations élémentaires de la planification de déploiement

Cette section décrit deux opérations élémentaires que nous avons identifiées et qui sont utiles pour simplifier l'implémentation de planificateurs de déploiement. Ces opérations sont réutilisables quel que soit l'algorithme de planification : l'une sert à vérifier la compatibilité entre un fichier binaire et un nœud de calcul, et l'autre permet de vérifier si deux nœuds de calcul sont capables de communiquer entre eux.

8.2.2.1 Compatibilité entre un binaire et un nœud de calcul

L'opération qui consiste à vérifier si un fichier binaire compilé (exécutable, DLL, *etc.*) de la description générique d'application est compatible avec un nœud de calcul de la description des ressources est commune à tous les planificateurs de déploiement.

Les paramètres à prendre en compte sont le système d'exploitation (son nom, son format binaire 32 bits ou 64 bits, sa version) et la nature l'architecture matérielle. Cette opération doit supporter la notion de compatibilité ascendante entre les binaires et les machines. Par exemple, un binaire compilé et optimisé pour une architecture `i486` peut s'exécuter sur une architecture `i686`, mais la réciproque n'est pas forcément vraie.

8.2.2.2 Communication entre des nœuds de calcul

L'opération qui consiste à vérifier si deux nœuds de calcul de la description des ressources sont capables de communiquer entre eux est une opération factorisable des planificateurs de déploiement.

Cette vérification doit aussi permettre de savoir par quels moyens les deux nœuds de calcul peuvent communiquer, c'est-à-dire par quelles technologies réseau, quels protocoles de communication, s'il y a des pare-feux, des passerelles de traduction d'adresses (NAT) ou des passerelles d'accès aux réseaux sous adresses IP privées, *etc.*

Suivant le modèle de description de la topologie réseau que nous avons présenté au chapitre 6, deux nœuds de calcul peuvent communiquer entre eux si et seulement s'il possèdent un groupe réseau parent (directement ou non) autre que le groupe fictif « *root vertex* ».

8.2.3 Discussion

Comme toute application se résume à un ensemble de processus et de codes à charger avec des contraintes de co-localisation et de communication, il est raisonnable d'envisager de planifier le déploiement d'applications de différents types avec un seul et même planificateur (pour un algorithme de planification donné). Le *bas niveau d'abstraction de la description générique d'applications* convient bien au travail du planificateur, qui doit placer les constituants de l'application sur des ressources qui sont décrites dans un formalisme de *bas niveau elles aussi* : la description générique d'une application est plus directement projetable sur la description des ressources que les descriptions spécifiques d'applications. La connaissance du paradigme de programmation n'est d'aucune aide pour le planificateur.

Au contraire, la description générique d'applications rend les planificateurs de déploiement plus faciles à implémenter, puisqu'ils n'ont pas à tenir compte des notions spécifiques aux différents types d'applications (*process-collocations*, *host-collocations*, ports de connexion, gestionnaires de connexion, *etc.*), et que GADe décrit *explicitement tous* les éléments à déployer. L'introduction de la description générique d'applications revient à *transférer un peu d'intelligence du planificateur vers les convertisseurs de descriptions d'applications* (spécifique vers générique). Bien qu'ils soient relativement simples, les convertisseurs allègent la tâche du planificateur en décrivant les applications dans des termes proches des ressources d'exécution.

8.3 Conclusion

Le modèle de description générique d'applications (GADe) décrit dans ce chapitre contient toutes les informations utiles au planificateur de déploiement pour qu'il puisse remplir son rôle de sélection et de placement.

L'introduction du concept de description générique d'applications visait initialement à factoriser les opérations de planification. Puis nous avons observé que ce concept permet aussi de factoriser certaines opérations de la phase d'exécution du plan de déploiement. Ainsi, nous avons enrichi la description générique d'applications avec des informations (telles que la définition de variables d'environnement, les répertoires courants d'exécution des programmes, *etc.*) qui ne sont pas utiles au planificateur de déploiement, mais dont l'exécuteur du plan a besoin.

L'enrichissement de la description générique d'applications permet effectivement de factoriser certaines opérations de la phase d'exécution du plan de déploiement, car le plan hérite directement de la description générique d'applications : il établit des associations entre des entités de la description générique d'applications et des nœuds de la description des ressources.

Pourquoi ne pas décrire les applications directement dans le formalisme de la description générique d'applications ?

- La description générique d'applications ne contient pas toute l'information d'une description spécifique. Cette perte d'information (lors de la conversion de spécifique vers générique) concerne essentiellement la *configuration* de l'application lors de la phase d'exécution du plan de déploiement. Le chapitre suivant revient sur ces questions de configuration, qui doit à nouveau utiliser les descriptions spécifiques d'applications.
- La description générique d'applications est plus compliquée à produire par le développeur, car tout doit être rendu explicite, et GADe ne s'exprime dans les termes propres au type de l'application qui rendent compte de sa structure : la description générique d'applications est un formalisme de bas niveau d'abstraction.

Ainsi, il est important de conserver les descriptions spécifiques d'applications.

Enfin, les contributions de ce chapitre ne sont pas spécifiques aux grilles de calcul, elles sont valables aussi dans d'autres environnements d'exécution. Simplement les grilles ont été la source d'inspiration de ce résultat dans le sens où les grilles de calcul peuvent accueillir des types d'applications très variés.

Chapitre 9

Configuration et adaptation des applications en fonction de leur environnement d'exécution

Sommaire

Le chapitre 6 a montré comment il est possible de décrire des topologies réseau complexes et aux *performances variées* (réseaux de clusters haute performance, réseaux locaux, réseaux longue distance, *etc.*). Puis le chapitre 7 a mis en évidence que des applications parallèles telles que MPI acceptent d'être *partitionnées*, afin d'exploiter une puissance de calcul supérieure en

	niveau de communication	latence typique
1	threads d'un processus ou processus d'une machine - *SMP*	~ 100 ns[1]
2	machines reliées par Myrinet - *SAN*	~ 3 μs
3	machines d'un réseau local Gigabit Ethernet - *LAN*	~ 100 μs
4	sites d'un même état - *WAN continental*	~ 10 ms à ~ 100 ms
5	sites sur deux continents distincts - *WAN intercontinental*	~ 350 ms

TAB. 9.1 – Latences typiques de communication.

s'exécutant sur un plus grand nombre de machines. L'objectif de ce chapitre est de montrer comment des applications parallèles MPI ou MVP peuvent s'adapter à la hiérarchie des performances de communication des infrastructures d'exécution, et comment l'outil de déploiement peut configurer ces applications pour qu'elles aient connaissance de la topologie réseau sous-jacente.

Nous commençons par rappeler les niveaux de la hiérarchie de performances réseau qui existent dans les grilles de calcul pour en déduire l'intérêt des implémentations hiérarchiques qui s'adaptent aux infrastructures d'exécution. Puis, nous montrons en détails comment des applications parallèles MPI et des systèmes de MVP peuvent tenir compte de la topologie réseau pour améliorer leurs performances d'exécution. Enfin, nous montrons comment l'outil de déploiement peut configurer ces applications automatiquement pour qu'elles aient connaissance de la topologie réseau sous-jacente.

9.1 Nécessité de l'adaptation à la hiérarchie de performances des grilles

9.1.1 Hiérarchie de performances dans les grilles de calcul

Comme l'a mis en évidence le chapitre 3, les grilles de calcul sont constituées de ressources très diverses, en particulier en ce qui concerne les capacités de communication. Il existe plusieurs ordres de grandeur pour les temps de communication :

1. entre deux threads d'un processus ou entre deux processus d'une même machine SMP ;
2. entre deux machines reliées par un réseau haute performance au sein d'un cluster (SAN) ;
3. entre deux clusters dans un même réseau local (LAN) ;
4. entre deux sites d'un même état (WAN continental) ;
5. et entre deux continents distincts (WAN intercontinental).

Cette hiérarchie de performances de communication intervient aussi bien dans le débit que dans la latence. À titre d'exemple, les latences typiques de communication sont données dans le tableau 9.1. Cependant, les avancées technologiques améliorent le débit plus facilement que la latence. Par exemple, le débit peut être virtuellement augmenté à l'infini en agrégeant les liens réseau, mais la latence restera toujours bornée par les temps de commutation (pour les technologies réseau qui utilisent ce mécanisme) et par la distance (pour toutes les technologies). Pour cette raison, les sections 9.2 et 9.3 s'attachent essentiellement à *masquer les latences* élevées dans les communications avec les machines les plus éloignées.

Implicitement, dans tout ce chapitre, nous supposons que les performances de communication sont constantes tout au long de l'exécution de l'application. Or force est de reconnaître que les performances des réseaux dans les grilles de calcul peuvent être très variables, puisque ces environnements sont partagés par un grand nombre d'utilisateurs. Cependant, cette hypothèse est valide à gros grain : il est peu probable que la hiérarchie de performances indiquée dans le tableau 9.1 s'inverse durablement pendant l'exécution. C'est d'autant plus vrai qu'en pratique, comme le montrent les sections suivantes,

- les *valeurs absolues* de débit ou de latence ne sont pas prises en compte, mais seulement leurs *rapports* entre les différents niveaux de communication ; par exemple, le rapport des latences entre les niveaux 2 et 3 du tableau 9.1 est de 33 ;
- ce ne sont pas les *valeurs précises* des rapports de débits ou de latences qui sont considérées, mais seulement une *classification qualitative* des niveaux de communication (« les communications de niveau 2 sont plus rapides que celles de niveau 3 »).

9.1.2 Applications plates *versus* applications hiérarchiques

Des travaux [135, 30, 100] de la communauté des grilles de calcul ont montré que nombre d'applications peuvent être optimisées pour s'exécuter plus rapidement sur des infrastructures distribuées aux performances de communication hétérogènes. C'est le cas en particulier des applications parallèles (MPI) et à mémoire partagée (MVP).

Pour optimiser ces applications, il est nécessaire de prendre en compte la hiérarchie de performances de la topologie réseau sous-jacente sur laquelle elles s'exécutent. Le principe général de ces optimisations consiste à *minimiser les communications sur les réseaux les plus lents* (en termes de latence ou de débit), et à *initier ces communications lentes le plus tôt possible*.

Enfin, ces optimisations « hiérarchiques » peuvent se faire soit au niveau du code source de l'application écrite par l'utilisateur, soit au niveau des bibliothèques ou des environnements d'exécution utilisés par l'application. L'idéal serait bien sûr que l'adaptation à la topologie réseau soit complètement transparente pour l'utilisateur, et donc qu'elle n'apparaisse pas dans le code source, mais les résultats de recherche dans ce domaine sont encore trop récents pour que toutes les bibliothèques de communication et tous les supports exécutifs soient implémentés de manière hiérarchique.

9.1.3 Discussion

En plus de la performance des réseaux de communication, nous remarquons qu'il pourrait également être intéressant de tenir compte de l'hétérogénéité des performances des nœuds de calcul et de stockage. Ce type d'information est déjà exploité en matière d'ordonnancement de tâches, mais il serait utile d'étudier comment une application concurrente peut s'adapter aux caractéristiques de calcul et de stockage des nœuds.

Les deux sections suivantes présentent brièvement comment, au cours de travaux antérieurs, nous avons rendu la bibliothèque MPICH-G2[2] système de MVP DSM-PM2[3] capables de s'adapter à la topologie réseau sous-jacente.

[1] ns : nanoseconde (10^{-9} seconde).

[2] Travaux réalisés en 2001 et 2002 sous la responsabilité de Nicholas KARONIS, à Argonne National Laboratory, USA, dans le groupe DSL (*Distributed Systems Laboratory*) dirigé par Ian FOSTER.

[3] Travaux réalisés en stage de DEA, sous la responsabilité de Gabriel ANTONIU et Luc BOUGÉ à l'IRISA.

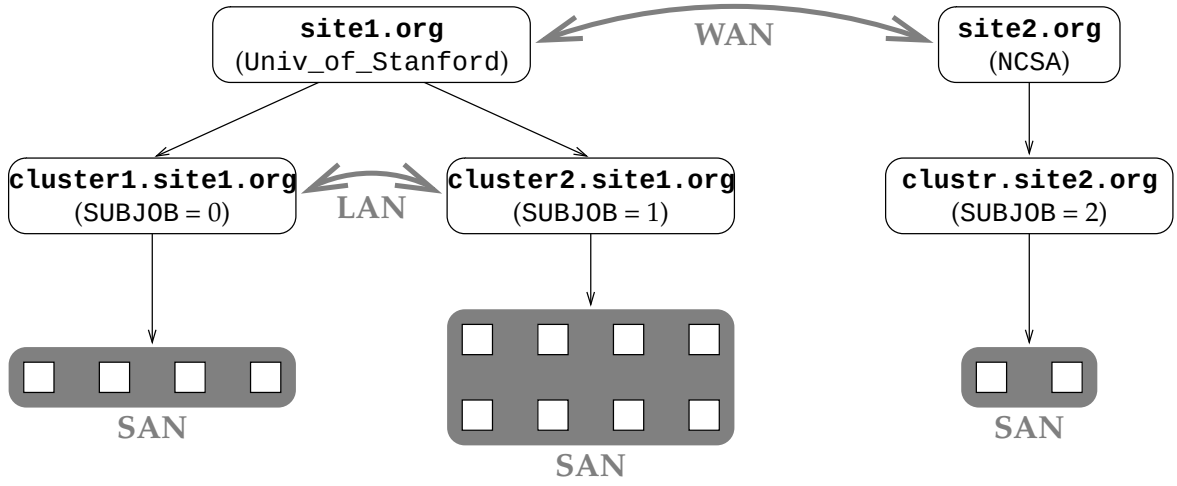


FIG. 9.1 – Représentation schématique dans MPICH-G2 de la topologie réseau correspondant à l'exemple de la figure 4.4.

9.2 Prise en compte de la topologie réseau dans MPICH-G2

La section 4.2.2.1 (page 59) a montré que les bibliothèques MPI MPICH-G2, PACX-MPI et MagPIe pouvaient être configurées avec la description de la topologie réseau sous-jacente sur laquelle l'application s'exécute. Cette section décrit de quelle manière cette information peut être exploitée, afin d'améliorer les performances d'exécution des programmes parallèles.

9.2.1 Accès à la topologie réseau sous-jacente

De plus en plus d'algorithmes de calcul parallèle savent s'adapter à la hiérarchie des performances de communication des infrastructures d'exécution [68]. Il est donc nécessaire de permettre à l'utilisateur d'avoir accès à la description de la topologie réseau depuis le code source. Par exemple, les tâches de calcul peuvent être partitionnées dans un programme MPI par la création de différents « communicateurs », *i.e.* des groupes de processus.

Le script RSL de soumission de tâches de la figure 4.4 (page 61) donne lieu à la configuration schématisée sur la figure 9.1. Dans cet exemple, tous les processus sur un même cluster possèdent une même valeur pour la variable `GLOBUS_DUROC_SUBJOB_INDEX`. Les processus des clusters au sein du même réseau local ont leur variable `GLOBUS_LAN_ID` fixée à une unique valeur. Implicitement, les sites avec des valeurs de `GLOBUS_LAN_ID` différentes sont interconnectés par des réseaux longue distance. Chaque processus traduit localement cette information en un « tableau de couleurs », c'est-à-dire sans avoir besoin de communiquer avec les autres processus MPI. Un tel tableau de couleurs est associé à chaque communicateur : c'est une description de la topologie réseau qui affecte, pour chaque niveau de communication, une « couleur » à chaque processus du communicateur. Les processus qui ont la même couleur à un niveau de communication donné peuvent communiquer entre eux par le moyen de communication de ce niveau. Par exemple, les processus d'un même réseau local ont la même couleur pour le niveau 3, mais ils peuvent avoir des couleurs différentes pour le niveau 2 s'ils se trouvent dans des clusters distincts.

```
int main (int argc, char *argv[])
{
   int my_rank, flag;
   MPI_Comm new_comm;
   int **colors;

   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   MPI_Attr_get(MPI_COMM_WORLD, MPICHX_TOPOLOGY_COLORS, &colors, &flag);
   if ( flag == 0 )
   {
      /* colors not available */
   }
   MPI_Comm_split(MPI_COMM_WORLD, colors[my_rank][2], 0, &new_comm);
   /* use "new_comm" for communications within my cluster */
   ...
}
```

FIG. 9.2 – Création d'un communicateur par cluster (SAN) dans MPICH-G2.

Pour chaque processus, un tableau de couleurs est attaché à chaque communicateur par le mécanisme « *attribute caching* »[4]. Les communicateurs peuvent avoir des différentes tailles, et ils peuvent inclure plusieurs réseaux locaux et plusieurs clusters. À chaque fois qu'un nouveau communicateur est créé, un tableau de couleurs est calculé localement et associé au communicateur nouvellement créé.

Nous avons implémenté, dans MPICH-G2, le mécanisme qui permet au développeur d'accéder à la description de la topologie réseau par le biais des attributs des communicateurs. La figure 9.2 montre un extrait de code C qui permet à l'utilisateur de créer dynamiquement un communicateur **`new_comm`** par cluster (niveau 2, d'où l'indice `[2]` dans le code). Ce nouveau communicateur MPI peut être utilisé par l'algorithme numérique pour partitionner les tâches de calcul, en maximisant les communications à l'intérieur de **`new_comm`**, et en évitant autant que possible les communications d'un cluster à un autre.

Ce mécanisme est intégré à MPICH-G2, qui est distribué sur le web en « *open source* »[5]. Il est par exemple utilisé par un algorithme d'écoulement pour simuler les artères du corps humain [68].

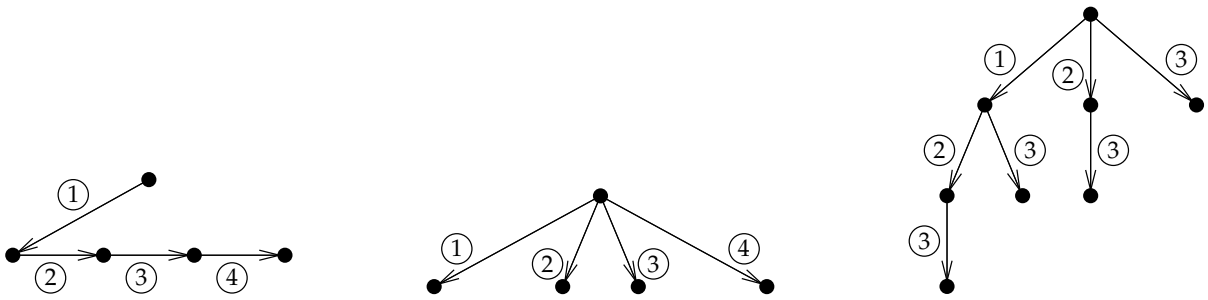
9.2.2 Opérations collectives hiérarchiques

La section précédente a montré comment MPICH-G2 permet au développeur d'accéder dynamiquement à la topologie réseau sous-jacente pour optimiser son algorithme hiérarchique de calcul. Ce mécanisme est très flexible, mais il n'est pas transparent. Cette section présente brièvement comment les opérations collectives MPI sont implémentées de manière hiérarchique dans MPICH-G2, pour une utilisation transparente du point de vue du développeur.

Les opérations collectives de MPI (*cf.* section 2.3.1.3, page 23) sont massivement utilisées par les algorithmes de calcul numérique. Elles font intervenir tous les processus d'un com-

[4] *Attribute caching* : mécanisme qui permet d'associer des données (ou attributs) à un communicateur MPI (conforme aux normes MPI).

[5] MPICH-G2 : `http://www.Globus.org/mpi/`.



(a) Chaînage. (b) À plat. (c) Arbre binomial.

FIG. 9.3 – Exemples de schémas de communication possibles pour les opérations collectives MPI.

municateur, qui peut s'étendre sur plusieurs clusters, voire plusieurs réseaux locaux répartis à travers différents continents. De nombreux algorithmes existent [154] pour implémenter les opérations collectives à partir de messages point à point (`MPI_Send` et `MPI_Recv`) : le choix d'un algorithme plutôt qu'un autre dépend souvent de la taille du message échangé par l'opération collective, du nombre de processus qui participent à l'opération (*i.e.*, le nombre de processus du communicateur), le débit et la latence de communication entre les processus, *etc.* La figure 9.3 illustre quelques schémas de communication possibles pour diffuser un message d'un processus vers tous les autres (`MPI_Bcast`). Ces algorithmes ont été initialement étudiés sur des infrastructures plates, où les performances de communication entre chaque paire de processus sont homogènes.

Les fédérations de clusters présentent deux niveaux de hiérarchie pour les performances réseau : les communications intra-clusters et les communications inter-clusters. MagPIe [108] permet d'adapter les schémas de communications aux fédérations de clusters. Par exemple, pour la diffusion (*BroadCast*), un algorithme de diffusion plat (*cf.* figure 9.3(b)) peut être utilisé pour la diffusion entre les clusters (réseau lent), et un algorithme de diffusion binomial (*cf.* figure 9.3(c)) peut être plus performant pour la diffusion entre les processus d'un même cluster (réseau rapide). Cette hiérarchisation des communications permet en outre de minimiser les communications sur les réseaux les plus lents.

Les travaux de Nicholas Karonis *et al.* [100] ont généralisé ces résultats aux grilles de calcul, avec un nombre quelconque de niveaux de communication (le tableau 9.1 énumère cinq niveaux de communication). À chaque niveau de communication, un algorithme est utilisé en fonction de la latence et du débit des communications à ce niveau. Nous avons implémenté onze opérations collectives[6] [9, 8, 12] de manière hiérarchique dans MPICH-G2 en tenant compte de quatre niveaux de communication (WAN, LAN, SAN, SMP).

Les trois opérations collectives de MPI-1 restantes n'ont pas été implémentées de manière hiérarchique (`MPI_Gatherv`, `MPI_Scatterv`, `MPI_Alltoallv`). En effet, dans les algorithmes efficaces comme celui de l'arbre binomial (*cf.* figure 9.3(c)), certains processus servent de relais : ils reçoivent des données qui ne leur sont pas destinées, mais qu'ils doivent faire suivre à d'autres processus. Or dans les trois opérations collectives restantes, les processus

[6] `MPI_Barrier`, `MPI_Bcast`, `MPI_Gather`, `MPI_Scatter`, `MPI_Reduce`, `MPI_Allgather`, `MPI_Allgatherv`, `MPI_Alltoall`, `MPI_Allreduce`, `MPI_Reduce_scatter`, `MPI_Scan`.

ne connaissent pas la taille des données qu'ils doivent relayer vers les autres processus, car chaque message à une taille arbitraire en fonction du processus destinataire. Donc ils ne peuvent pas jouer leurs rôles de relais. Le seul moyen pour que tous les processus connaissent la taille des données qu'ils vont devoir faire suivre serait de procéder à une première phase de diffusion des tailles des messages. Mais le coût de cette diffusion préliminaire annihilerait complètement le bénéfice de l'implémentation hiérarchique de l'opération collective.

9.2.3 Discussion

Les opérations collectives MPI sont encore souvent pensées en termes d'envoi et de réception de messages *point à point*. Cependant, à l'intérieur de certains réseaux locaux ou certains clusters, il serait intéressant d'utiliser les possibilités de *multicast* pour implémenter les opérations collectives.

Les choix d'algorithmes pour les opérations collectives à chaque niveau de communication sont souvent fondés sur les *rapports de latence* seulement. Il serait intéressant de considérer également les *rapports de débits* entre chaque niveau de communication. Enfin, les premiers pas décrits ici vers les implémentations hiérarchiques sont fondés sur une *classification qualitative* entre les niveaux de communication. Cependant, il pourrait être utile de choisir les algorithmes en fonction des valeurs *numériques* des latences et débits, en plus des nombres de processus et des tailles de messages.

9.3 Gestion hiérarchique des verrous de synchronisation dans un système de MVP

La section précédente a présenté deux mécanismes qui permettent aux applications MPI de tirer parti de l'hétérogénéité des performances réseau : le premier laisse la responsabilité au développeur d'adapter son algorithme à la topologie réseau, le deuxième est transparent pour l'utilisateur. Cette section présente un mécanisme transparent de synchronisation pour les systèmes de MVP multi-threads. L'objectif général reste le même, à savoir minimiser les communications les plus lentes au bénéfice des communications plus rapides, afin d'améliorer les performances d'exécution.

Luciana Arantes *et al.* [25] ont proposé un protocole de cohérence[7] entre les copies répliquées d'un espace d'adressage virtuellement partagé : CLRC (*Clustered Lazy Release Consistency*) vise à améliorer la *localité des données* sur des fédérations de clusters, et a été implémenté dans le système de MVP TreadMarks [102, 20]. Suivant cette approche, les clusters possèdent tous des *proxies* qui stockent les modifications (« *diffs* ») apportées aux pages sur lesquelles les processus du cluster local écrivent. Ainsi, des accès consécutifs à la même page par des nœuds du même cluster peuvent réutiliser les *diffs* stockés dans le proxy local du cluster.

L'approche que nous présentons dans cette section ne vise pas à améliorer la localité des données, mais à tirer parti de la *localité dans la gestion des verrous*. En effet, les verrous de synchronisation sont massivement utilisés dans les systèmes de MVP qui implémentent des modèles de cohérence relâchée. Dans un tel modèle, tous les accès aux variables globales de

[7] Protocole et modèle de cohérence : *cf.* section 2.3.3.1, page 27.

l'espace d'adressage virtuellement partagé doivent avoir lieu entre deux points de synchronisation, tels que des barrières ou bien à l'intérieur d'une section critique (tant pour l'écriture que pour la lecture), protégée par exemple par un verrou.

Nous commençons par présenter le protocole de cohérence plat sur lequel nous illustrons nos travaux, puis nous montrons comment nous l'avons rendu hiérarchique grâce au concept de « libération partielle de verrou ».

9.3.1 Protocole de cohérence plat

Le relâchement du modèle de cohérence permet d'obtenir de meilleures performances en minimisant le nombre d'échanges de messages entre nœuds : il est par exemple inutile de propager à tous les nœuds les multiples écritures sur une même variable partagée faites au cours d'une section critique ; il suffit de transmettre la dernière valeur écrite avant la sortie de section critique.

Le protocole de cohérence HBRC (*Home-Based Release Consistency* [166, 97]) adhère au modèle de la cohérence relâchée : la mémoire n'est rendue cohérente qu'aux points de synchronisation. Dans ce protocole, chaque page partagée est attachée statiquement à un « nœud hôte » tout au long de l'exécution : ce nœud hôte héberge la copie de référence de la page (un nœud peut héberger les copies de référence de plusieurs pages partagées). L'hôte d'une page a toujours les droits en lecture et écriture sur cette page.

Ce protocole autorise les lecteurs et écrivains multiples[8] : des threads concurrents sur des nœuds distincts peuvent modifier en même temps des parties différentes d'une même page. Ces modifications se font à l'intérieur de sections critiques. À l'occasion d'un point de synchronisation (par exemple une sortie de section critique), un nœud qui a modifié localement une page envoie ses modifications (*diffs*) au nœud hôte de la page qui les applique sur sa copie de référence, puis invalide toutes les copies répliquées de cette page. Afin d'assurer la cohérence de la mémoire, le thread qui libère le verrou et qui a envoyé ses *diffs* au nœud hôte ne peut pas céder le verrou à un autre thread tant que tous les accusés de réception d'invalidation n'ont pas été reçus. Ensuite, un accès à une page invalidée provoque une interruption de « défaut de page » : dans ce cas, la page à jour est rapatriée dans son intégralité depuis le nœud hôte de cette page.

9.3.2 Limitations du protocole plat

Dans une configuration hiérarchique de grille telle que celle illustrée sur la figure 9.4, nous avons identifié deux facteurs qui limitent la performance du protocole HBRC plat.

1. Les implémentations distribuées de verrous de synchronisation ne prennent pas en compte les questions de localité pour décider de l'ordre dans lequel les candidats à l'obtention d'un verrou peuvent entrer section critique [141, 163]. Ainsi, il se peut qu'un verrou soit souvent échangé entre deux clusters, entraînant une perte de temps lors du transfert du verrou sur le réseau lent entre les deux clusters.

[8] MRMW : *Multiple Readers, Multiple Writers*.

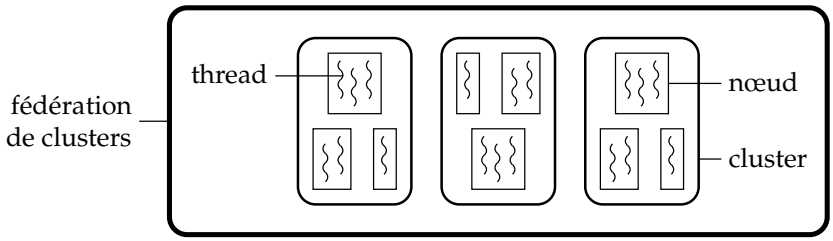


FIG. 9.4 – Hiérarchie des performances de communication prise en compte dans DSM-PM2.

2. Lors d'une invalidation de page par son nœud hôte, les accusés de réception proviennent de tous les nœuds qui possèdent une copie répliquée de la page : ces nœuds peuvent se trouver dans le même cluster ou bien dans un autre réseau local. Ainsi, le délai d'attente des accusés de réception est limité inférieurement par les latences des communications inter-clusters, et il retarde d'autant le transfert du verrou au thread suivant qui le réclame.

Les sections suivantes décrivent deux mécanismes qui permettent de vaincre ces limitations pour améliorer les performances d'exécution d'un système de MVP au modèle de cohérence relâchée.

9.3.3 Gestion hiérarchique des verrous

Sur l'exemple de la figure 9.5, le nœud N_1 du cluster A acquiert un verrou L en le demandant au gestionnaire de ce verrou. Ensuite, le nœud N_3 du cluster B réclame ce même verrou auprès du gestionnaire, qui fait suivre la requête vers le nœud N_1 qui possède le verrou. Un peu plus tard, le nœud N_2 du cluster A demande ce verrou auprès du gestionnaire qui fait suivre la requête vers N_1. Au moment où le nœud N_1 veut sortir de section critique et libérer le verrou (« *release* »), un protocole de gestion non hiérarchique donnerait le verrou au nœud N_3 d'abord, et ensuite seulement au nœud N_2, conformément à l'ordre d'arrivée des requêtes de verrou. Cette approche donne lieu à deux communications lentes entre les clusters A et B pour échanger le verrou.

Dans le but de limiter le nombre de messages échangés aux niveaux de communication les plus lents, notre protocole de gestion des verrous tient compte de la topologie réseau sous-jacente en donnant la *priorité aux threads les plus proches* (en termes de performances de communication) pour transmettre les verrous. Ainsi, dans l'exemple de la figure 9.5, N_1 cède le verrou au nœud N_2 qui se trouve dans le même cluster, bien que la requête de N_2 soit arrivée en dernier. N_2 est informé que le verrou est censé aller ensuite vers le nœud distant N_3, à moins qu'une requête en provenance du cluster A n'arrive avant la libération du verrou par N_2. Cette approche minimise le nombre de communications entre les clusters, pour favoriser les échanges plus rapides au sein des clusters.

Ce mécanisme revient à ré-ordonner les acquisitions de verrous, mais il ne viole pas la sémantique de ces objets de synchronisation. Ce mécanisme est reproduit à l'identique entre les threads de chaque nœud : le verrou est transféré de préférence à un thread qui s'exécute dans le même nœud, plutôt que vers un autre nœud (du même cluster ou d'un autre cluster). Ainsi,

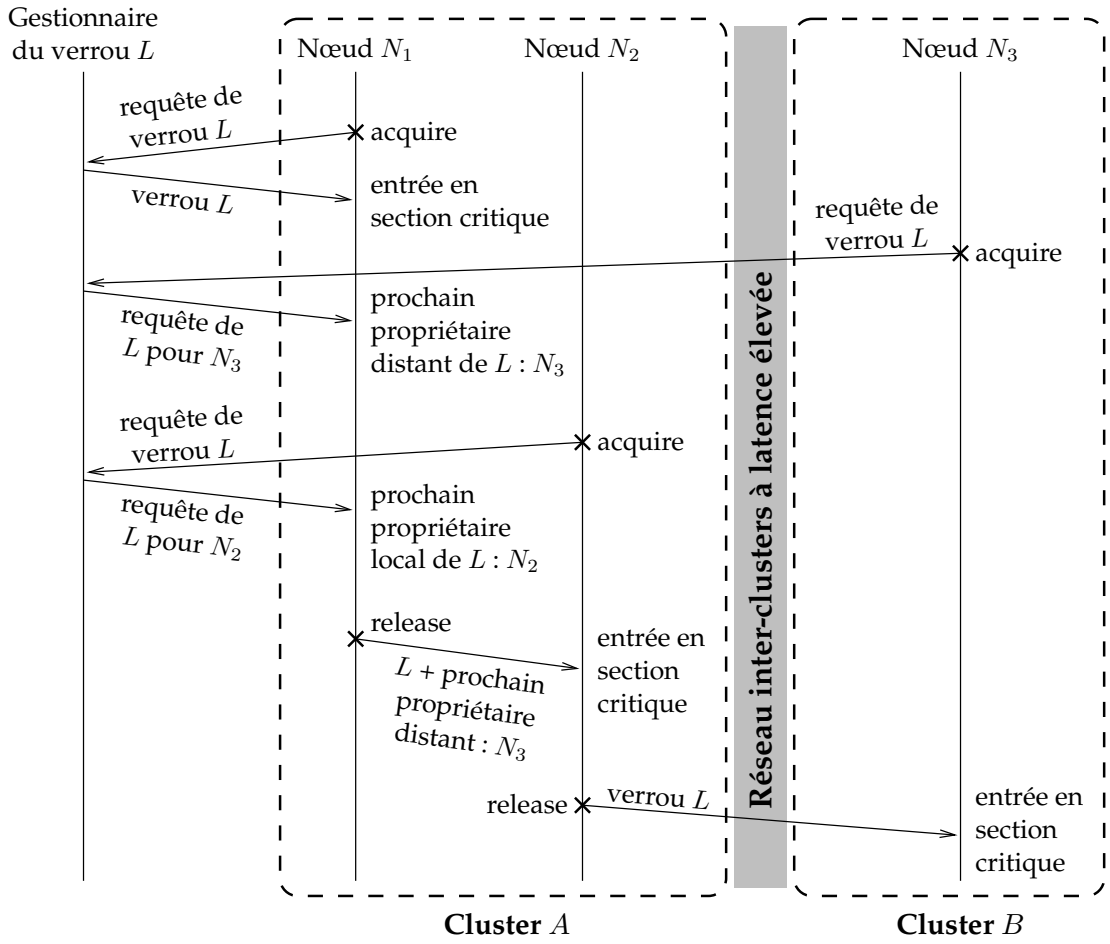


FIG. 9.5 – Illustration de la gestion hiérarchique des verrous de synchronisation dans DSM-PM2.

la gestion hiérarchique des verrous répond à la première limitation listée à la section 9.3.2 ; la section 9.3.6 ci-dessous lève la deuxième limitation.

9.3.4 Minimisation des envois de *diffs*

Dans la version plate du protocole HBRC, à la libération d'un verrou, un thread envoie *systématiquement* les *diffs* aux nœuds hôtes des pages qu'il a modifiées. Si un thread sort de section critique et transmet le verrou à un thread qui s'exécute dans le *même processus*, alors les *diffs* ne sont pas envoyés tout de suite aux nœuds hôtes. Les modifications seront transmises seulement lorsque le verrou quittera le nœud.

La gestion hiérarchique des verrous se conjugue avec ce mécanisme pour limiter plus encore le nombre de messages échangés sur le réseau. En effet, la situation où un thread transmet un verrou à un thread du même processus est rendue plus fréquente grâce à la plus grande priorité accordée aux threads locaux pour céder les verrous.

Ce mécanisme de retardement de la transmission des *diffs* est correct du point de vue de la cohérence mémoire car deux threads d'un même processus partagent le même espace d'adressage : les modifications faites par un nombre quelconque de threads d'un processus qui ont acquis successivement le même verrou sont agrégées et envoyées en une seule fois aux nœuds hôtes.

9.3.5 Contrer le manque d'équité

La gestion hiérarchique des verrous de synchronisation donne la préférence aux threads locaux et aux nœuds locaux : si deux threads sont en compétition pour acquérir le verrou, c'est toujours le thread le plus proche qui l'emporte. Ce ré-ordonnancement des attributions des verrous est inéquitable, et il peut conduire à des situations de famine (*starvation*). Pour contrer ce manque d'équité, il est impératif de définir une limite sur le nombre maximal d'acquisitions consécutives d'un verrou par les threads d'un même nœud ; il faut également limiter le nombre d'acquisitions consécutives d'un verrou par les nœuds d'un même cluster. Ces limites peuvent être spécifiées au système de MVP par le biais de variables d'environnement.

Ainsi, lorsqu'un verrou a été acquis N_{max} fois consécutives par les nœuds d'un même cluster, et s'il est réclamé dans un autre cluster, alors le mécanisme de priorité aux nœuds locaux ne s'applique pas, et le verrou est forcé de changer de cluster. Si $N_{max} = 1$, alors la gestion des verrous est non hiérarchique. Si $N_{max} = \infty$, alors l'équité n'est plus assurée, et il y a risque de famine. En d'autres termes, la détermination des limites N_{max} permet de troquer l'équité pour la performance d'exécution, sans sacrifier la correction des résultats, et sans conduire à des *livelocks*.

9.3.6 Libération partielle de verrou

Dans la version plate du protocole HBRC, lors de la libération d'un verrou, un thread envoie des *diffs* aux nœuds hôtes. Avant de libérer *effectivement* le verrou et le transmettre à un autre thread, pour ne par risquer d'accès mémoire incohérents, le thread qui sort de section critique doit attendre d'avoir reçu :

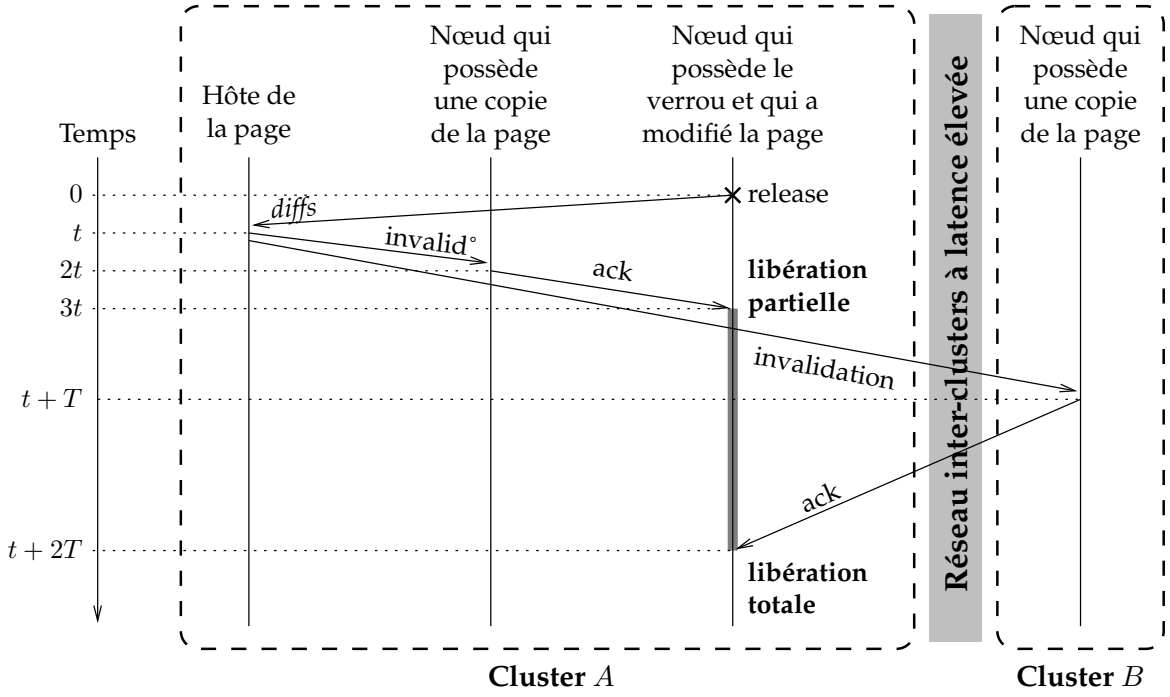


FIG. 9.6 – Temps d'arrivée des accusés de réception d'invalidation (ack) selon les clusters ; T est le temps de transfert d'un message entre deux clusters ; t est le temps de transfert d'un message à l'intérieur d'un cluster.

- les accusés de réception des *diffs* par les nœuds hôtes, qui attestent de la prise en compte des modifications ;
- et les accusés de réception des invalidations de pages de la part des nœuds qui possédaient une copie répliquée des pages modifiées.

Or les accusés de réception d'invalidation en provenance des nœuds du cluster local arrivent généralement avant ceux en provenance des clusters distants, comme l'illustre la figure 9.6. En effet, les liens réseau intra-clusters offrent une meilleure performance que les liens entre les clusters.

Pour tirer profit de cet ordre d'arrivée des accusés de réception, nous introduisons le concept de « *libération partielle de verrou* » (*partial release*). Un verrou est *partiellement* libéré lorsque tous les accusés de réception d'invalidation de pages *en provenance du cluster local* ont été reçus. Un verrou est *totalement* libéré lorsque *tous* les accusés de réception d'invalidation *en provenance de tous les clusters* ont été reçus. Ainsi, un verrou partiellement libéré peut être transmis à un nœud du cluster local sans risque d'accès mémoire incohérent. Un verrou ne peut migrer d'un cluster à un autre que s'il est totalement libéré.

Nous remarquons qu'un verrou partiellement libéré peut être transféré plusieurs fois de nœud en nœud au sein d'un même cluster sans qu'aucun accusé de réception ne soit reçu des clusters distants. En revanche, le verrou ne peut quitter le cluster que lorsque tous les accusés de réception d'invalidation ont été reçus.

Comme sur la figure 9.6, nous notons t le temps de transfert d'un message entre deux nœuds d'un même cluster, et T le temps de transfert d'un message entre deux clusters dis-

tincts. En première approximation, nous supposons que les performances réseau entre les clusters sont homogènes, et nous faisons l'hypothèse que les temps de transfert (t ou T) ne changent pas, que le message soit un *diff*, un ordre d'invalidation ou un accusé de réception. C'est raisonnable si les pages sont peu modifiées, car les *diffs* sont alors des messages de petite taille, à l'instar des invalidations et des accusés de réception. La figure 9.6 illustre le cas le plus favorable : le nœud hôte se trouve dans le même cluster que le verrou en cours de libération (cluster A). Le gain de temps est alors $(2T+t)-(3t) = 2(T-t)$. Dans le cas le moins favorable, le nœud hôte se trouve dans le cluster distant qui attend le verrou (cluster B) : le gain est alors nul $(2T+t)-(2T+t)$. Dans le cas intermédiaire, le plus probable si l'application s'exécute sur plus de trois clusters, le nœud hôte ne se trouve dans aucun des clusters A et B, et le gain est de $(3T)-(2T+t) = T-t$. Statistiquement, le gain de temps moyen est $T-t \gg 0$, puisqu'il y a au moins un ordre de grandeur entre les latences de deux niveaux de communication consécutifs (*cf.* tableau 9.1). Une évaluation expérimentale du gain de performance est rapportée dans le chapitre 11 (page 189).

Nous appliquons le mécanisme de libération partielle de verrou à l'échelle des clusters. Cependant, nous pourrions l'appliquer avec une granularité plus fine. Un verrou peut être transmis à *n'importe nœud dont l'accusé de réception a été reçu*, qu'il s'agisse d'un nœud du cluster local ou d'un cluster distant. Cependant, nous partons du principe que les temps de d'envoi de messages sont quasiment homogènes entre les clusters. Nous observons alors une première vague d'accusés de réception en provenance des nœuds du cluster local, suivie d'une deuxième vague en provenance des clusters distants. Il n'est donc pas utile d'alourdir les structures de données liées à la gestion des verrous : nous pouvons nous contenter de simplement faire la distinction entre les nœuds proches d'une part (ceux du cluster local), et les nœuds distants d'autre part.

9.3.7 Discussion

Cette section a présenté deux mécanismes hiérarchiques pour améliorer la performance d'exécution de systèmes de MVP fondés sur un modèle de cohérence relâchée, *i.e.* avec des points de synchronisation. Ces mécanismes augmentent la *localité dans la gestion de la synchronisation*, par opposition à la *localité des données*. Nous avons l'intuition que ces deux approches sont complémentaires : il serait intéressant d'étudier comment les deux mécanismes visant à favoriser la localité peuvent se combiner.

Les barrières sont également des points de synchronisation : il serait certainement intéressant de reprendre les travaux réalisés dans le cadre des barrières MPI pour les appliquer aux systèmes de MVP. Les travaux de recherche sur MPI ont aussi montré qu'il est utile de tenir compte d'un plus grand nombre de niveaux de hiérarchie : il pourrait être utile de mettre ce résultat en application pour les systèmes de MVP.

Par définition de l'opération de barrière[9], le concept de libération partielle de verrou ne peut pas être appliqué aux barrières. Cependant, les *sémaphores*, les *moniteurs* et les *verrous* sont des mécanismes de synchronisation équivalents : chacun de ces mécanismes peut être implémenté à l'aide de n'importe quel autre. Ainsi, il serait intéressant d'étudier comment les sémaphores et les moniteurs peuvent tirer parti du mécanisme de libération partielle de verrou.

[9]Un thread ne peut quitter une barrière que lorsque tous les autres threads participant à l'opération ont atteint la barrière.

Pour pouvoir exploiter ces mécanismes, le système de MVP doit avoir accès à l'information sur la topologie réseau sous-jacente. Comme l'illustre la figure 4.6 (page 63), la configuration de la MVP DSM-PM2 se fait par le biais d'un fichier qui décrit les liens réseau. La section suivante montre comment un outil de déploiement automatique peut procéder à cette configuration sans intervention de l'utilisateur qui veut déployer son application.

9.4 Configuration automatique des applications

Comme nous l'avons vu à la section 5.4 (page 92), l'exécution du plan de déploiement comporte diverses phases :

1. transfert et installation des fichiers (données et exécutables) sur les ressources indiquées dans le plan de déploiement ;
2. lancement des processus avec un environnement correctement configuré : support exécutif, variables d'environnement, fichiers de configuration, répertoire de travail, *etc.* ;
3. et configuration de l'application après lancement des processus : chargement des DLL dans les serveurs de conteneurs CCM, valeurs initiales des attributs de composants, connexions de leurs ports, activation de l'application avec `ORB->run()`, *etc.*

Cette section montre comment l'outil de déploiement peut configurer automatiquement l'application en cours de lancement au moment de l'exécution du plan de déploiement.

9.4.1 Opérations génériques et opérations spécifiques

Comme l'a évoqué la section 5.4.4 (page 93), les trois phases mentionnées ci-dessus ne sont pas nécessairement indépendantes les unes des autres, et elles peuvent s'entrelacer. De plus, nous avons vu à la section 8.3 (page 157) que le modèle de description générique d'applications (GADe) a été étendu pour pouvoir factoriser une parties des opérations d'exécution du plan de déploiement.

Ainsi, comme l'illustre la figure 9.7, les informations contenues dans le plan de déploiement et dans la description générique d'applications permettent de réaliser certaines opérations communes pour tous les types d'applications : définition de variables d'environnement, transferts de fichier, soumission de tâches à distance, placement dans le répertoire d'exécution, *etc.* En revanche, certaines opérations spécifiques à chaque type d'application ne sont pas factorisées, telles que la création de fichiers de configuration, la définitions de certaines variables dont la valeur n'est pas connue statiquement (mais seulement après la planification), la génération des tâches à soumettre (commandes pour lancer les programmes avec leurs arguments en ligne de commande), *etc.*

Pour effectuer les opérations spécifiques de configuration d'une application, il est nécessaire de revenir à sa description spécifique. Le retour à la description spécifique du type d'application en cours de déploiement est toujours possible : grâce à l'élément `specific_appl_descr` (*cf.* section 8.1, page 138), GADe permet de spécifier des liens entre chacun des objets du modèle de la description générique d'applications (groupes de processus, processus, codes à charger, connexions, implémentations) et les objets de la description spécifique d'application (composants CCM, programmes MPI, *etc.*).

Opérations génériques (indépendantes du type d'application)	**Opérations spécifiques au type d'application en cours de déploiement (plugin)**
Transfert et installation de fichiers	Création de fichiers de configuration

`#!/bin/sh`
 	
`ENV_VAR="its_value"`
`export ENV_VAR`
 	
`cd /working/directory/`
	`CLUSTER="Rennes"`
	`export CLUSTER`
	
	`./ComponentServer --ior ./cs.ior \`
	`  -ORBInitRef NameService="IOR:..."`

Soumission de la tâche sur une ressource distante et récupération de sa sortie Rapport de déploiement	analyse de la sortie (coordonnées, erreurs, *etc.*)

FIG. 9.7 – Exemple d'exécution du plan de déploiement pour une application CCM.

Par exemple, pour une application GRIDCCM, une tâche à soumettre peut être un composant parallèle MPI, c'est-à-dire plusieurs processus qui se lancent avec la commande `mpiexec`. Ainsi, lors de l'exécution du plan, un module spécifique à chaque type d'application est responsable de créer la ligne de commande correspondant à chaque tâche. Ensuite, l'exécuteur du plan de déploiement peut lancer la commande spécifique sur la ressource distante sélectionnée pour cette tâche, de manière générique, c'est-à-dire sans recourir à des opérations spécifiques au type d'application en cours de déploiement.

De même, pour les applications CCM, si un service de nommage est nécessaire, sa référence (IOR) doit être transmise aux serveurs de conteneurs comme le montre la figure 9.7 : cette opération est spécifique au type d'application en cours de déploiement.

9.4.2 Plugins spécifiques des types d'applications

Cette *coopération* entre les opérations génériques et spécifiques de l'exécution du plan de déploiement peut être réalisée sous la forme de plugins. Tout type d'application supporté par l'outil de déploiement doit définir un ensemble de fonctions spécifiques qui sont invoquées par l'exécuteur générique du plan.

Comme l'illustre la figure 9.7, ces fonctions spécifiques sont la création de fichiers de configuration avant que l'exécuteur générique ne les transfère, la définition de variables d'environnement spécifiques qui dépendent du placement des programmes par exemple, la génération de la ligne de commande avec ses arguments pour la tâche à soumettre, l'analyse de la sortie de cette commande afin de générer le rapport de déploiement avec les coordonnées des tâches lancées (PID, IOR, *etc.*) et avec les éventuels messages d'erreur, *etc.*

C'est de cette manière que l'outil de déploiement peut transmettre l'information sur la topologie réseau aux applications MPICH-G2 en fixant les variables d'environnement, ou encore au système de MVP DSM-PM2 en générant un fichier de configuration tel que celui de la figure 4.6 (page 63).

Enfin, le plugin spécifique d'un type d'application doit pouvoir appeler des fonctions du plugin d'un autre type d'application. C'est utile par exemple pour les applications mixtes GRIDCCM : si un composant a une implémentation parallèle fondée sur MPI, alors la génération de la commande de lancement de la tâche GRIDCCM correspondant à ce composant parallèle peut se faire en appelant la fonction de création de commande du plugin d'applications MPI.

9.4.3 Discussion

Cette section a présenté le mécanisme de plugin qui permet à l'exécuteur générique du plan de déploiement d'intégrer les opérations spécifiques à chaque type d'application. En examinant d'autres types d'applications, nous n'excluons pas qu'il soit nécessaire d'étendre les plugins à d'autres fonctions, qui interviendraient sur d'autres opérations de l'exécution du plan de déploiement. La section 10.1.3.4 (page 183) énumère les plugins que nous avons implémentés dans notre outil de déploiement.

9.5 Conclusion

Ce chapitre a commencé par rappeler les différents niveaux de la hiérarchie des performances réseau des grilles de calcul, et il a souligné la nécessité d'adapter les applications à leur environnement d'exécution pour obtenir les hautes performances chères aux applications parallèles. Puis il a montré comment des applications MPI ou celles fondées sur la MVP peuvent s'adapter à l'hétérogénéité des performances de communication, afin de diminuer les temps d'exécution de ces applications parallèles. Enfin, ce chapitre a décrit comment un outil de déploiement peut configurer automatiquement ces applications pour qu'elles aient connaissance de la topologie réseau sous-jacente : par un mécanisme de plugins, l'exécuteur générique du plan de déploiement coopère avec des fonctions spécifiques aux différents types d'applications.

À l'avenir, il pourrait être intéressant de considérer les débits (ou d'autres caractéristiques) en plus de la latence pour rendre les applications hiérarchiques. Il pourrait même être utile de passer d'une classification qualitative (réseaux « lents » *vs.* réseaux « rapides ») à des valeurs numériques pour décrire les caractéristiques réseau, telles que les rapports de latences, de débits, *etc.* Pour terminer, si les propriétés de l'environnement changent durablement et sensiblement, il pourrait être intéressant d'adapter dynamiquement les applications.

Comme l'a montré le chapitre 3, l'hétérogénéité ne concerne pas seulement les liens réseau, mais également les puissances de calcul et de stockage des nœuds. Pour les applications qui savent s'adapter à ces paramètres, il serait également utile d'enrichir les plugins spécifiques de configuration.

Troisième partie

Mise en œuvre et évaluation

Chapitre 10

Mise en œuvre au sein d'ADAGE

Sommaire

Les chapitres précédents de ce document ont présenté en détails notre proposition pour automatiser le déploiement d'applications sur des grilles de calcul. L'objectif de ce chapitre est de valider nos résultats, en décrivant leur mise en œuvre dans un outil de déploiement baptisé ADAGE.

Après une description d'ensemble d'ADAGE, nous présentons chacun de ses modules : les descriptions spécifiques des applications et leur conversion en description générique, la découverte des ressources, la planification, et l'exécution du plan de déploiement.

10.1 Présentation générale d'ADAGE

10.1.1 Contexte

ADAGE est l'acronyme de « *Automatic Deployment of Applications in a Grid Environment* ». Cet outil de déploiement automatique fait partie du projet Padico [65, 216], un environnement

logiciel pour la programmation des grilles de calcul. Padico vise le calcul haute performance avec la programmation parallèle, distribuée (composants logiciels), et mixte (composants parallèles). En particulier, Padico s'intéresse aux applications de couplage de codes fondées sur le concept de composants CORBA parallèles. Padico est constitué des sous-systèmes suivants :

PaCO++ est une infrastructure mettant en œuvre le concept d'objets CORBA parallèles ;

GRIDCCM met en œuvre le concept de composants CORBA parallèles ;

PadicoTM (Padico *Task Manager*, le gestionnaire de tâches Padico) est la plate-forme de communication haute performance utilisée par PaCO++ et GRIDCCM ;

ADAGE est l'outil de déploiement automatique.

10.1.2 Considérations générales de conception

Dans son état actuel, ADAGE est constitué d'environ 21 000 lignes de code C++. Ce code est portable dans le sens où aucune primitive système de bas niveau n'est utilisée, ni aucun code assembleur. ADAGE dépend de diverses bibliothèques :

- OpenLDAP 3 ou Globus MDS2 pour l'accès aux informations sur les ressources des serveurs LDAP du MDS de Globus ;
- Globus GASS pour rapatrier des fichiers à distance via différents protocoles de transfert (GridFTP, FTP, HTTP, HTTPS) ; sinon la commande `wget` est utilisée, mais elle ne supporte par le protocole GridFTP ;
- ZZIPlib pour l'extraction des fichiers contenus dans les packages d'applications (archives compressées ZIP) ; sinon, la commande `unzip` est utilisée ;
- Xerces-C++ 2.6 pour l'analyse des documents XML ;
- Pathan 1.2 ou Xalan-C++ 1.9 pour la recherche de données dans les documents XML (DOM[1] et XPath) ;

ADAGE dépend aussi de la commande `ssh` et éventuellement du module de gestion des ressources de Globus (client GRAM) pour la soumission de tâches à distance. Enfin, ADAGE peut dépendre de la commande `scp` pour le transfert de fichiers.

10.1.3 Architecture générale d'ADAGE

Les objectifs d'ADAGE sont d'être *simple* d'utilisation et de supporter des types d'applications variés, tels que MPICH-G2, MPICH1-P4, JXTA, CCM, GRIDCCM. Cette section décrit l'architecture générale de notre outil de déploiement pour supporter ces différents types d'application, puis elle montre la simplicité de l'interface utilisateur d'ADAGE.

10.1.3.1 Structuration en modules

Comme l'illustre la figure 10.1, nous avons structuré le code d'ADAGE en plusieurs modules : description (spécifique et générique) des applications, description des paramètres de contrôle, découverte des ressources, planification, et exécution du plan de déploiement.

Nous remarquons que dans cette architecture, les paramètres de contrôle sont fournis par l'utilisateur sous la forme d'un fichier XML, puis analysés (chargés en mémoire). Comme l'illustre la figure 10.2, ces paramètres de contrôle interviennent dans tous les modules.

[1] DOM : *Document Object Model*, une interface d'accès et de manipulation en mémoire de documents XML.

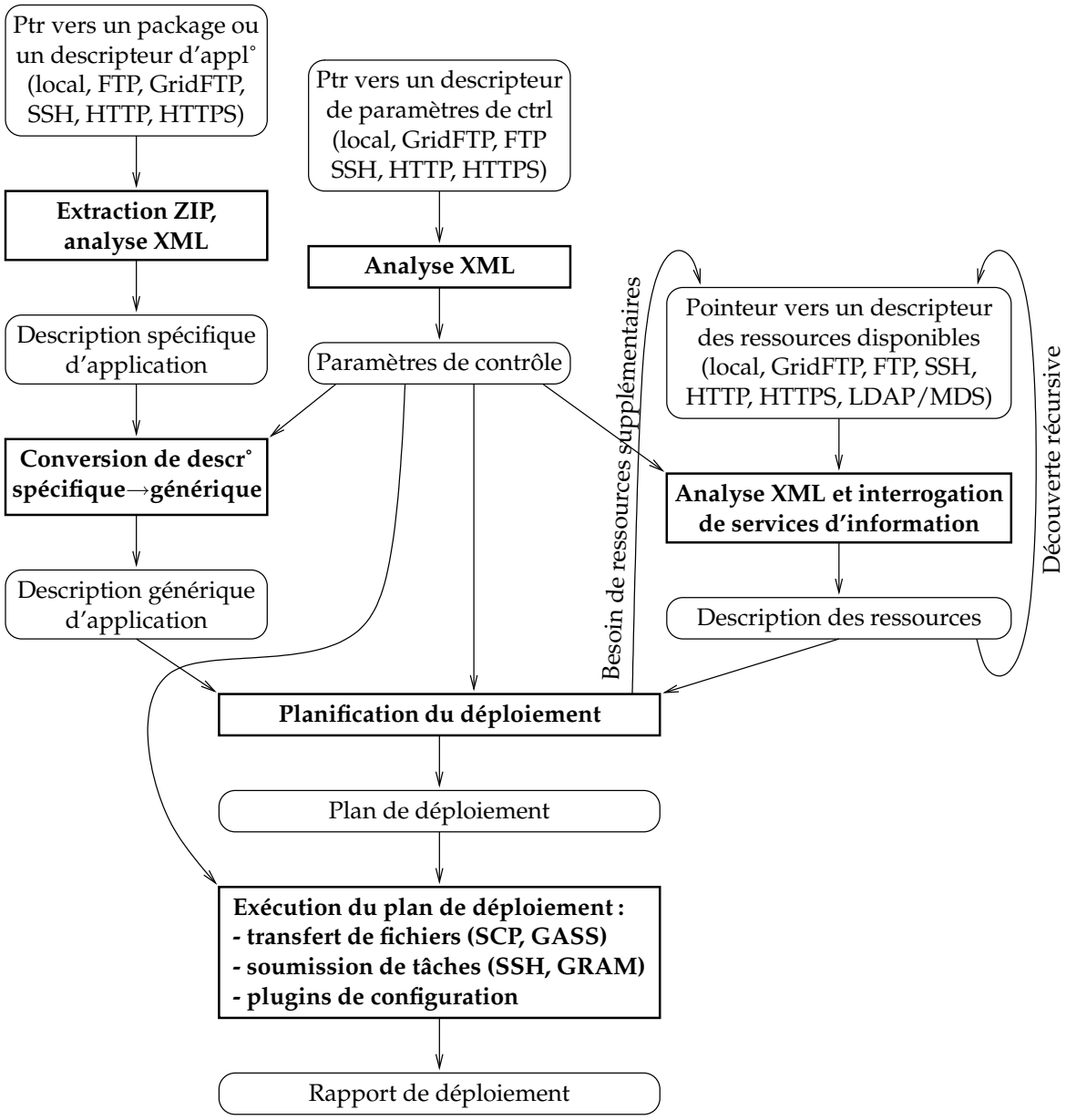


FIG. 10.1 – Vue d'ensemble de l'architecture d'ADAGE.

```
<ctrl_params>
   <!-- Pour la conversion de description spécifique vers générique -->
   <CCM_Component_Server>
      http://www.irisa.fr/paris/ADAGE/ComponentServer.zip
   </CCM_Component_Server>
   <Corba_Naming_Service>
      http://www.irisa.fr/paris/ADAGE/NamingService.zip
   </Corba_Naming_Service>
   <cardinality id="the_component_id"> 7 </cardinality>
   <!-- Pour la planification de déploiement -->
   <planner>random</planner>
   <!-- Pour l'exécution du plan de déploiement -->
   <ssh username="ux455083"/>
</ctrl_params>
```

FIG. 10.2 – Exemple de fichier XML décrivant des paramètres de contrôle.

Conversion de description spécifique d'application en description générique : pour spécifier par exemple où ADAGE peut trouver les exécutables du serveur de conteneur (*ComponentServer*) ou du service de nommage, packagés comme de simples composants (*cf.* section 8.1.3.2, page 146), ou encore pour imposer la cardinalité d'un composant CCM, d'un processus MPI, *etc.*

Découverte des ressources disponibles de la grille : par exemple pour filtrer les ressources de sites à proscrire, ou imposer de « n'utiliser que les machines du domaine **`irisa.fr`** », *etc.*

Planification : par exemple pour indiquer l'algorithme de planification à utiliser (*random* ou *round-robin*), ou bien pour spécifier le nombre maximal de sites sur lesquels l'application peut être partitionnée, *etc.*

Exécution du plan de déploiement : par exemple pour spécifier le nom d'utilisateur à employer si le protocole SSH est utilisé, *etc.*

10.1.3.2 Interface utilisateur

Conformément à son objectif de simplicité, ADAGE se présente sous la forme d'une unique commande intégrée, qui s'utilise par exemple de la manière suivante :

```
deploy -appl http://www.applications.fr/linux/fft/fast.zip
       -res ldap://mds.grid.org/mds-vo-name=local,o=grid
       -ctrl_params my_control_params.xml
```

ADAGE offre plusieurs points d'entrée :

– une description *spécifique* d'application[2], éventuellement incluse dans un package, ainsi qu'un pointeur vers la description des ressources disponibles[3], et éventuellement un pointeur vers la description des paramètres de contrôle[2] ;

[2]Fichier local, ou bien descripteur à rapatrier par GridFTP, FTP, SCP, HTTP, ou HTTPS.

[3]Fichier local, ou bien descripteur à rapatrier par GridFTP, FTP, SCP, HTTP, HTTPS, ou encore LDAP/MDS.

– une description *générique* d'application[2], au format XML, ainsi qu'un pointeur vers la description des ressources disponibles[3], et éventuellement un pointeur vers la description des paramètres de contrôle[2] ;
– un plan de déploiement[2], au format XML, à exécuter.

ADAGE offre également différents points de sortie possibles :

– après conversion d'une description spécifique d'application en une description générique, ADAGE peut produire en sortie la description générique de l'application au format XML, et s'arrêter avant la planification de déploiement ;
– après découverte des ressources, ADAGE peut produire en sortie la description des ressources disponibles au format XML, et s'arrêter avant la planification de déploiement ;
– après la planification, ADAGE peut produire en sortie le plan de déploiement au format XML, et s'arrêter avant de l'exécuter ;
– après exécution du plan de déploiement, ADAGE produit un rapport de déploiement.

10.1.3.3 Flexibilité

La multitude des points d'entrée et de sortie d'ADAGE permet à l'utilisateur de vérifier et de modifier manuellement les documents intermédiaires (description générique d'application, plan de déploiement, description des ressources), pour les fournir en entrée à ADAGE. C'est utile notamment à des fins de mise au point : normalement, les interventions de l'utilisateur se font par le biais des paramètres de contrôle.

10.1.3.4 Plugins spécifiques des applications

Chaque type d'application supporté par ADAGE donne lieu à un plugin qui déclare les fonctions correspondant aux opérations spécifiques du processus de déploiement (*cf.* section 9.4.1, page 172) : conversion de la description spécifique en description générique d'application, création de fichiers de configuration (optionnel), génération de commandes à lancer (avec variables d'environnement, et arguments en ligne de commande) pour soumettre les tâches, analyse de la sortie de cette commande pour alimenter le rapport de déploiement, *etc.* Ces opérations correspondent à des méthodes virtuelles dans le code C++.

Les plugins actuellement intégrés dans ADAGE permettent de déployer des applications CCM, MPICH1-P4, MPICH-G2, GRIDCCM, JXTA[4]. Cependant, le plugin CCM est incomplet : nous n'avons pas implémenté les opérations de connexions des ports et la définition des valeurs initiales des attributs des composants. OpenCCM (*cf.* section 4.3.3.3, page 67) permet de réaliser ces opérations automatiquement, à partir de la description spécifique de l'application CCM : il pourrait être intéressant d'intégrer cette implémentation dans le plugin CCM d'ADAGE.

10.1.4 Discussion

L'architecture générale d'ADAGE que nous venons de présenter reflète bien l'architecture de l'outil de déploiement introduite à la section 5.1.2.1 (page 79). Sa flexibilité offre plusieurs

[4]Remerciements à Mathieu JAN pour le plugin JXTA.

points d'entrée et de sortie, pour vérifier par exemple les choix faits par le planificateur de déploiement avant d'effectivement lancer l'application. ADAGE prouve la possibilité de simplifier le déploiement grâce à l'automatisation pour des applications de types variés : parallèles, distribuées, mixtes, pair-à-pair.

Une limitation réside dans la description des paramètres de contrôle : les contraintes permettant de contrôler les divers aspects du déploiement automatique sont regroupées dans un seul et unique descripteur, qui intervient dans tous les modules d'ADAGE. Il pourrait être intéressant de rationaliser l'expression des paramètres de contrôle, en les classant en différentes catégories. Il pourrait aussi être utile de supporter une gradation des niveaux d'exigence des paramètres de contrôle, qui peuvent être plus ou moins impératifs.

10.2 Description spécifique des applications et conversion en description générique

10.2.1 Description spécifique d'application

En entrée, ADAGE accepte à la fois des descripteurs d'application (au format XML) et des packages d'applications (au format ZIP) qui contiennent dans le répertoire **`meta-inf/`** un descripteur de l'application. ADAGE déduit le type du fichier (descripteur XML ou package ZIP) et le type de l'application (CCM, MPI, JXTA) à partir des extensions de fichiers. Les descripteurs de composants (CCM et GRIDCCM) ont pour extension **`.csd`**, et leurs packages **`.zip`**. Les descripteurs d'assemblages de composants (CCM et GRIDCCM) ont pour extension **`.cad`**, et leurs packages **`.aar`**. Les descripteurs d'applications MPI (MPICH1-P4 et MPICH-G2) ont pour extension **`.mpi`**, et leurs packages **`.mpizip`**. Les descripteurs d'applications JXTA ont pour extension **`.jxta`**.

Il serait plus rationnel, à l'avenir, qu'ADAGE reconnaisse le type de fichier automatiquement sans se fier à l'extension, grâce à la bibliothèque **`libmagic`** par exemple. De plus, ADAGE devrait pouvoir deviner le type d'application en analysant seulement l'entête des descripteurs XML.

Les plugins spécifiques des types d'applications sont responsables d'analyser (« *parsing* ») les descripteurs XML pour les charger en mémoire, sous forme DOM avec Xerces-C++ (*cf.* section 10.1.2).

10.2.2 Conversion en description générique d'application

Les plugins spécifiques des types d'applications sont chargés de convertir les descriptions spécifiques en descriptions génériques d'applications. Comme l'illustre la figure 10.1, ces convertisseurs prennent, en entrée, la description spécifique d'application et les paramètres de contrôle. Ils fournissent en sortie la description générique au format DOM en mémoire.

Pour les applications dont la description est relativement simple, telles que MPI ou JXTA, les convertisseurs de description spécifique comptent environ 400 lignes de code C++ chacun. En revanche, pour les applications CCM et GRIDCCM de structure plus complexe, et dont la description est répartie dans plusieurs descripteurs, le convertisseur compte environ 1 200 lignes de C++.

10.2.3 Bilan

Avec les bibliothèques de manipulation de documents XML, le module d'analyse des applications est relativement léger. Autant que faire se peut, nous avons utilisé le modèle DOM pour représenter les documents en mémoire, ce qui facilite leurs transformations et la recherche de données. Nous avons écrit les DTD[5] des descriptions spécifiques d'applications MPI, GRIDCCM, JXTA, ainsi que celle de la description générique d'applications.

10.3 Découverte des ressources

Pour la description des ressources, nous avons mis en œuvre le modèle présenté dans le chapitre 6 avec le langage XML. Les descripteurs de ressources sont de simples fichiers, qui peuvent être rapatriés par les protocoles HTTP(S), (Grid)FTP, SCP, ainsi que depuis les serveurs LDAP du MDS de Globus2. Nous avons défini une entrée (binaire) dans la hiérarchie de l'arbre LDAP du MDS qui permet de stocker le fichier XML de description des ressources (encodé en `Base64` dans l'attribut `Mds-XML-Resource-Description`).

10.3.1 Distribution des informations

Le fichier spécifié en ligne de commande à ADAGE n'est qu'un point de départ : il peut contenir la description de quelques ressources de calcul et de communication, mais également des pointeurs vers d'autres descripteurs de ressources. Ainsi, comme l'illustre la figure 10.1, si le planificateur a besoin de plus de ressources que celles qui sont décrites dans le fichier indiqué en ligne de commande par l'utilisateur, alors il peut demander au module de découverte des ressources qu'il suive quelques liens vers des ressources supplémentaires au fur et à mesure des besoins du planificateur. Les descripteurs supplémentaires peuvent à leur tour référencer d'autres fichiers de description des ressources, de manière récursive. ADAGE vérifie qu'il n'y a pas de cycles dans les liens en mémorisant les localisations des descripteurs de ressources déjà rapatriés.

Cette distribution des informations participe aux propriétés de passage à l'échelle de notre modèle de description. De plus, elle simplifie leur maintenance, puisque chaque site qui met à disposition des ressources dans une organisation virtuelle de grille peut être responsable de mettre à jour la description de ses ressources.

Si le planificateur a besoin de connaître l'intégralité des ressources disponibles (pour calculer un plan de déploiement optimal par exemple, ce qui peut poser des problèmes de passage à l'échelle), alors il peut demander au module de découverte des ressources de suivre récursivement tous les liens de tous les fichiers de description des ressources.

10.3.2 Coopération entre les différents systèmes d'information

Les descripteurs de ressources supplémentaires référencés peuvent être rapatrié par n'importe quel protocole supporté par ADAGE (mentionnés dans la section précédente). Mais il serait également possible de rapatrier ces données à partir des services d'information d'UNICORE.

[5] DTD : *Document Type Definition*, description qui sert à valider la correction syntaxique des documents XML.

Il se peut, par exemple, que la fréquence d'un processeur ne soit pas spécifiée dans un descripteur XML de ressources. Or le MDS de Globus contient cette information de façon native, c'est-à-dire sans l'ajout de nos descripteurs de ressources. Comme nous l'avons décrit à la section 6.3.3 (page 114), l'élément `external_information` permet de compléter les informations sur les nœuds de calcul avec celles qui sont contenues dans le MDS de Globus (les données natives qui font partie du service d'information d'UNICORE pourraient également être utilisées).

10.3.3 Discussion

Grâce au mécanisme de coopération entre notre modèle de description des ressources et les différents systèmes d'information déjà existants, il n'est pas nécessaire de modifier ces systèmes d'information pour exploiter leurs données. En revanche, si l'administrateur d'un site décide de rendre la description des ressources suivant notre modèle accessible depuis un serveur MDS de Globus2, il doit modifier légèrement la configuration de MDS pour y intégrer un fichier XML. Cependant, le MDS modifié reste compatible avec les clients qui n'utilisent pas cette description des ressources suivant notre modèle.

L'intégration de la description des ressources sous la forme d'un fichier XML dans la hiérarchie d'un serveur LDAP nous fait perdre les fonctionnalités de filtrage de ce type de serveur. Mais en remplacement, nous bénéficions des capacités de filtrage et de recherche d'information dans des données XML avec XPath.

10.4 Planification et exécution du plan de déploiement

En entrée, les planificateurs de déploiement d'ADAGE prennent la description générique de l'application, la description des paramètres de contrôle, et une description d'une partie des ressources de la grille (point de départ). Toutes ces informations se trouvent en mémoire sous forme DOM, donc facilement exploitables avec la bibliothèque de recherche de données XML XPath. Le plan de déploiement produit est aussi du XML en mémoire sous forme DOM.

Dans ADAGE, nous avons développé deux planificateurs qui implémentent les deux algorithmes de planification présentés à la section 5.3.2.2 (page 89), à savoir *random* et *round-robin*. Le planificateur par défaut est *round-robin*, mais un autre planificateur peut être sélectionné par le biais des paramètres de contrôle (*cf.* figure 10.2). Chacun de ces deux planificateurs est capable de planifier le déploiement de n'importe quel type d'application, grâce à la notion de description générique d'applications.

Les deux planificateurs que nous avons implémentés sont simples : ils ne supportent que quelques paramètres de contrôle (limites sur le nombre de groupes en lesquels une application peut être partitionnée, *etc.*), et ils ne tiennent pas encore compte des contraintes de communication des applications (« connexions » dans GADe).

En l'état actuel, ADAGE supporte deux protocoles de soumission de tâches à distance (SSH et Globus/GRAM), et deux protocoles de transferts de fichiers (SCP et Globus/GASS). Il serait intéressant d'élargir la palette de protocoles supportés, en utilisant une bibliothèque telle que Elagi (*cf.* section 4.3.1.2, page 64).

Conformément au modèle décrit à la section 9.4 (page 172), l'exécuteur du plan de déploiement dans ADAGE fait intervenir les fonctions qui implémentent les opérations spécifiques au type d'application en cours de déploiement. Le rapport de déploiement dans ADAGE n'est rien d'autre que le plan de déploiement enrichi avec quelques éléments XML, qui désignent les fichiers déjà transférés, ainsi que les coordonnées des tâches lancées (PID, références Globus, IOR, *etc.*). Le rôle d'ADAGE s'arrête après la phase d'exécution du plan de déploiement.

10.5 Conclusion

Ce chapitre a décrit ADAGE, l'outil de déploiement automatique que nous avons mis en œuvre : il démontre la possibilité de simplifier le processus de déploiement d'applications complexes et de types variés sur des grilles de calcul, afin de le rendre accessible à n'importe quel scientifique non expert en informatique. La structure générale d'ADAGE répond point par point à l'architecture que nous avons introduite au chapitre 5.

ADAGE est un prototype de recherche et présente quelques limitations. En particulier, il ne gère par tout le cycle de vie des applications. Il fonctionne en mode « batch » : une fois que l'application est lancée, il retourne un rapport de déploiement qui contient toutes les informations nécessaires (PID, IOR, références Globus, *etc.*) pour contrôler l'exécution de l'application (la suspendre, interroger son état, la terminer prématurément, *etc.*). L'implémentation de ces fonctionnalités serait intéressante, notamment dans un environnement partagé de grilles, pour tuer les processus, et pour effacer les fichiers installés sur les ressources.

Chapitre 11

Évaluation des performances de l'implémentation hiérarchique du système de MVP DSM-PM2

Sommaire

La section 9.3 (page 165) a présenté quelques mécanismes pour améliorer les performances d'exécution des applications parallèles fondées sur la mémoire virtuellement partagée : la gestion hiérarchique des verrous de synchronisation, la libération partielle des verrous, et les limites pour éviter les situations de famine ont été mises en œuvre dans le système de MVP DSM-PM2 (*cf.* section 2.3.3.2, page 29). Ce chapitre présente quelques tests de performance qui mettent en évidence l'efficacité de chacun des mécanismes hiérarchiques que nous proposons, dans un environnement de fédérations de clusters.

La plate-forme expérimentale que nous avons utilisée est constituée de PC (Pentium II cadencés à 450 MHz, sous Linux 2.2), interconnectés par un réseau FastEthernet complètement commuté, et équipés de cartes réseau SCI relativement anciennes (type D310). Sur cette plate-forme, les latences inter-clusters observées sont de 100 μs (FastEthernet/TCP), et les latences intra-cluster sont de 8 μs (SCI/SISCI), soit un rapport de latences d'environ 12.

N_{max}^{thrd}	1	5	15	25	∞
$\frac{T_{flat}}{T_{hierarchy}}$	1	3.4	5.8	6.9	$\simeq$ 60 (inéquitable)

TAB. 11.1 – Impact de la limite sur le nombre d'acquisitions consécutives du verrou par les threads d'un même nœud.

11.1 Priorité des threads locaux au sein d'un nœud

Cette section évalue l'impact de la priorité accordée aux threads au sein d'un nœud pour transmettre un verrou.

11.1.1 Description du test

Le programme de test que nous utilisons s'exécute sur quatre nœuds d'un cluster SCI. Chaque nœud comporte quatre threads. Chaque thread entre et sort de section critique 10 000 fois en acquérant et libérant immédiatement toujours le même verrou. Les sections critiques sont vides : aucune opération n'est effectuée sur la mémoire partagée afin de ne mesurer que le gain de performance dû à la politique de priorité des threads locaux pour l'acquisition du verrou. Dans cette mesure, nous faisons varier le nombre limite N_{max}^{thrd} d'acquisitions consécutives du verrou par les threads d'un même nœud. Nous mesurons le temps T_{flat} pour exécuter l'application jouet avec la version plate du protocole HBRC ; $T_{hierarchy}$ est le temps d'exécution de l'application avec la gestion hiérarchique des verrous.

11.1.2 Résultats et discussion

Le ratio $\frac{T_{flat}}{T_{hierarchy}}$ mesure le gain de performance apporté par la gestion hiérarchique des verrous pour les threads au sein d'un nœud. Le tableau 11.1 montre que ce ratio augmente de manière spectaculaire lorsque N_{max}^{thrd} croît. En effet, plus N_{max}^{thrd} est élevé et plus les transferts de verrou à l'intérieur d'un nœud sont favorisés, et donc les échanges de verrou entre les nœuds diminuent. Même si SCI est un réseau très rapide, il est plus efficace de limiter le nombre de migrations de verrou d'un nœud à un autre.

11.2 Libération partielle de verrou

Cette section évalue l'impact des mécanismes combinés de libération partielle de verrou et de priorité aux nœuds locaux du cluster.

11.2.1 Description du test

L'application test utilisée s'exécute sur un nombre variable de clusters interconnectés par FastEthernet. Chaque cluster est composé de deux nœuds, interconnectés par SCI. Sur chaque

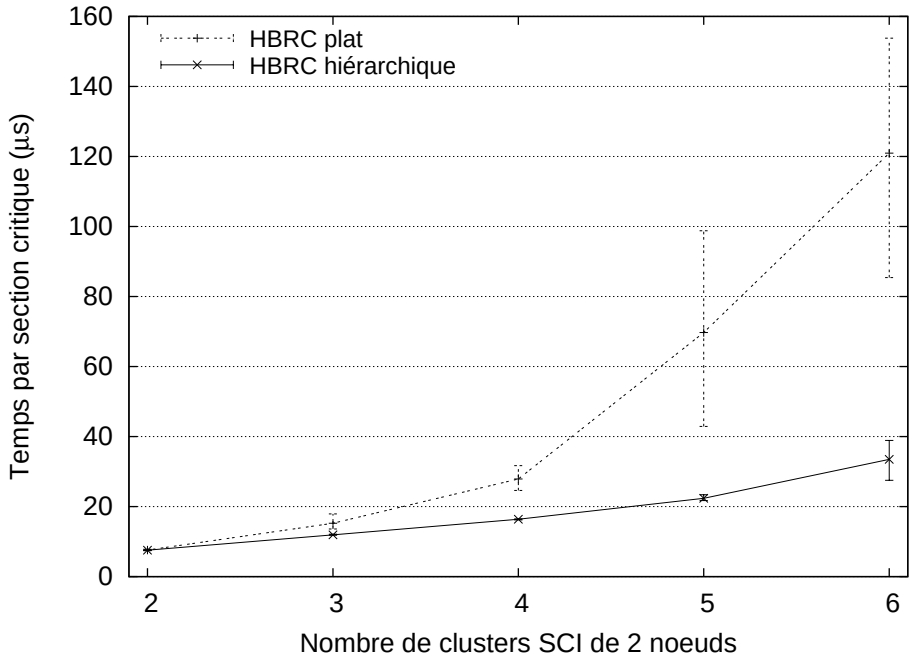


FIG. 11.1 – Impact de la libération partielle de verrou sur les performances d'exécution d'application avec protocole HBRC dans DSM-PM2.

nœud, un thread exécute 10 000 sections critiques : il acquiert un verrou, incrémente une unique variable entière partagée (4 octets), et il libère le verrou. Le même verrou est utilisé concurremment par tous les threads. Il y a un seul thread par nœud pour ne pas faire intervenir le mécanisme de priorité des threads locaux dans les mesures.

Nous mesurons le temps d'exécution de cette application, d'une part avec le protocole HBRC plat, donc en attendant *tous* les accusés de réception avant de transmettre le verrou, et d'autre part avec le protocole HBRC hiérarchique et la libération partielle de verrou. Ces mesures sont effectuées sans limite sur le nombre maximal d'acquisitions consécutives d'un verrou par les différents nœuds d'un même cluster ($N_{max}^{node} = \infty$) : c'est le cas le plus favorable, où le protocole n'impose pas l'équité.

11.2.2 Résultats et discussion

La figure 11.1 montre que le gain de performance est flagrant. Avec un rapport de latences de 12, le test s'exécute trois fois plus vite sur cinq clusters, et quatre fois plus vite sur six clusters. Le gain augmente avec le nombre de clusters : plus il y a de liens réseaux lents, et plus le protocole hiérarchique tire parti de la libération partielle de verrous. En effet, au fur et à mesure que le nombre de clusters croît, le protocole plat attend plus d'accusés de réception en provenance des clusters distants, tandis que le protocole hiérarchique évite d'attendre ces accusés de réception.

Cette observation est confirmée par la figure 11.2 qui montre que le nombre de messages inter-clusters constatés avec le protocole hiérarchique est bien inférieur à celui avec le proto-

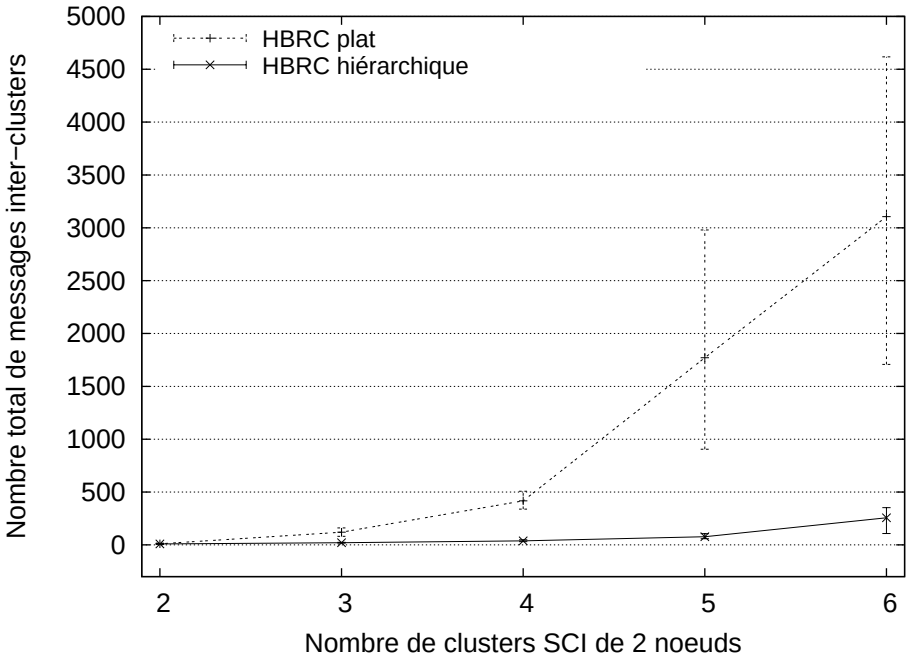


FIG. 11.2 – Impact de la libération partielle de verrou sur le nombre de messages inter-clusters avec protocole HBRC dans DSM-PM2.

cole plat. Au vu du nombre de messages inter-clusters, nous remarquons également que le verrou voyage plusieurs fois de cluster en cluster, alors que $N_{max}^{node} = \infty$. En effet, même si le protocole de gestion hiérarchique des verrous n'impose pas l'équité, il se peut que l'ordonnancement des requêtes laisse le verrou quitter un cluster pour y revenir plusieurs fois. Par exemple, si le verrou est réclamé par un nœud d'un cluster distant, et que les requêtes d'obtention du verrou en provenance du cluster local ne sont pas encore parvenues au gestionnaire, alors il va changer de cluster.

Enfin, les barres d'amplitude des figures 11.1 et 11.2 montrent que le protocole hiérarchique a un comportement beaucoup plus stable et régulier que le protocole plat (chaque mesure a été effectuée quatre fois). Les variances des temps d'exécution et des nombres de messages inter-clusters sont nettement plus faibles dans le cas du protocole hiérarchique, car l'ordonnancement des obtentions de verrou est contraint par la priorité accordée aux nœuds locaux, donc elle masque les effets liés à l'incertitude sur l'ordre d'arrivée des requêtes de verrou.

11.3 Minimisation des envois de *diffs*

Cette section évalue l'impact de la minimisation du nombre de *diffs* envoyés, en agrégeant les modifications faites par les threads d'un processus qui acquièrent successivement le même verrou. Comme l'a expliqué la section 9.3.4 (page 169), ce mécanisme bénéficie de la priorité accordée aux threads locaux d'un nœud pour la transmission des verrous.

N_{max}^{thrd}	1	5	15	25	∞
$\frac{T_{always}}{T_{aggregate}}$	1	2.1	4.7	7.3	(inéquitable)

TAB. 11.2 – Impact de l'agrégation des *diffs* dans le cas d'acquisitions consécutives du même verrou par les threads d'un processus.

11.3.1 Description du test

Pour ce test, nous utilisons la même configuration que celle décrite à la section 11.1.1. Quatre nœuds interconnectés par SCI comportent chacun quatre threads qui exécutent 10 000 sections critiques avec modification d'une variable partagée : un thread acquiert le verrou, incrémente une variable entière partagée (4 octets), et libère le verrou. T_{always} désigne le temps d'exécution de l'application avec le protocole HBRC hiérarchique, mais dont le mécanisme d'agrégation des *diffs* a été désactivé : ce protocole envoie *systématiquement* les modifications de pages à chaque libération de verrou, tout en bénéficiant de la priorité accordée aux threads locaux. $T_{aggregate}$ mesure le temps d'exécution de l'application avec le protocole hiérarchique, qui agrège les *diffs* pour ne les envoyer que lorsque le verrou quitte le nœud.

11.3.2 Résultats et discussion

Le ratio $\frac{T_{always}}{T_{aggregate}}$ donne le gain de performance apporté par l'agrégation des *diffs*. Le tableau 11.2 montre que ce ratio augmente sensiblement avec N_{max}^{thrd}. En effet, la minimisation du nombre de *diffs* envoyés fait également diminuer le nombre d'invalidations, et donc le nombre d'accusés de réception d'invalidation à attendre.

11.4 Conclusion

Ce chapitre a montré comment chaque mécanisme individuel améliore les temps d'exécution d'une application jouet en tenant compte des performances de communication entre les threads, suivant leur localisation. Il serait néanmoins intéressant de poursuivre ces expériences sur des applications réelles, telles que celles de la suite de tests « Splash-2 » [162]. Nous pensons que les applications qui utilisent massivement les verrous sur des sections critiques courtes pourraient tirer le meilleur parti des mécanismes hiérarchiques de synchronisation. Parmi les applications « Splash-2 », les programmes « *Ocean* » et « *Cholesky* » semblent les plus favorables. Ces expérimentations mériteraient également d'être menées à plus grande échelle.

Chapitre 12

Conclusion et perspectives

12.1 Conclusion générale

Contexte d'étude

Le *calcul scientifique* est un champ important de l'informatique : il permet par exemple de réaliser des économies sur l'étude de la déformation des structures des automobiles grâce à la simulation numérique. De nos jours, le calcul scientifique intervient dans des domaines toujours plus nombreux, tels que l'astrophysique, la modélisation climatique, l'économie, *etc.*

Les *applications de calcul scientifique* sont souvent particulièrement *complexes*, tant du point de vue de leur structure que de leur taille. Parmi les grandes classes d'applications de calcul scientifique, nous comptons les *applications distribuées* (à base de composants logiciels), les *applications parallèles* (par passage de messages ou par mémoire virtuellement partagée), et les *applications mixtes*, constituées de plusieurs composants qui peuvent être des programmes parallèles. Les applications à base de composants logiciels, tels que CCM, offrent la dynamicité, le support de l'hétérogénéité, la structuration des applications, et la réutilisabilité des codes. Les applications parallèles, fondées sur MPI ou sur les MVP, visent la haute performance en réduisant les temps d'exécution et en acceptant de plus grands volumes de données. Les composants parallèles, tels que GRIDCCM, sont idéaux pour les applications complexes de couplage de codes parallèles. Cependant, ces modèles de programmation offrent un support variable et limité au déploiement de leurs applications.

Ces applications présentent d'importants besoins en *puissance de calcul et de stockage*. Les *grilles informatiques* sont à même d'offrir la puissance nécessaire aux applications de calcul scientifique. Cependant, les grilles sont des environnements *complexes* : leurs ressources sont hétérogènes (ordinateurs et réseaux de communication), et elles sont distribuées géographiquement à grande échelle entre des sites qui se font une confiance limitée. Les intergiciels d'accès aux ressources des grilles fournissent des services de bas niveau, tels que la sécurité (chiffrement, authentification, *etc.*), des méthodes de transfert de fichiers et de soumission de tâches simples à distance.

Actuellement, les applications de calcul scientifique sont particulièrement difficiles à déployer sur des grilles. Le processus de déploiement est encore trop *manuel* et *techniquement complexe*. Bien que l'état de l'art en matière de déploiement soit riche, il ne permet pas encore

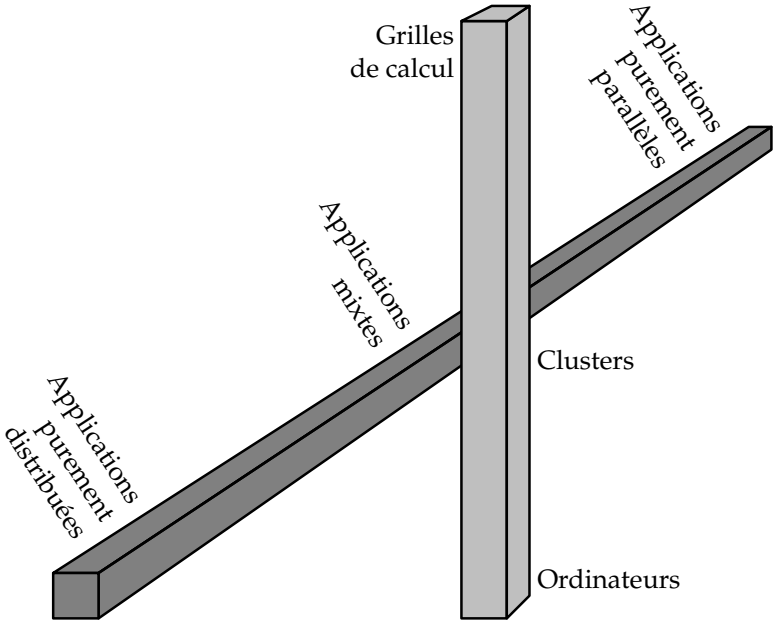


FIG. 12.1 – Deux mondes orthogonaux : les applications et les infrastructures d'exécution.

aux physiciens, chimistes, biologistes, *etc.* de déployer *simplement* leurs applications de calcul scientifique sur des grilles informatiques : l'utilisateur qui veut déployer son application doit faire preuve d'expertise dans le domaine du lancement des différents types d'applications, ainsi qu'en matière d'intergiciels d'accès aux ressources des grilles. De plus, l'utilisateur doit lui-même connaître les besoins de l'application, découvrir et sélectionner les ressources d'exécution nécessaires à l'application, choisir les implémentations de l'application pour chaque ressource sélectionnée, réaliser tous les transferts de fichiers et les lancements de tâches à distance, puis configurer l'application.

Contribution à l'automatisation du déploiement d'applications statiques

Dans le but de simplifier le déploiement d'applications statiques sur des grilles de calcul, nous avons choisi de masquer les difficultés de ce processus en l'*automatisant*, afin de le rendre accessible à des scientifiques non experts en informatique et en grilles.

Pour ce faire, nous sommes partis d'une double observation.

- D'une part, les modèles de programmation, de structuration et de déploiement des applications sont largement indépendants des infrastructures d'exécution et des intergiciels d'accès aux grilles de calcul. Par exemple, des applications MPI peuvent s'exécuter sur des ordinateurs multi-processeurs à mémoire partagée, sur des clusters, ou bien sur des grilles de calcul. De même, le modèle de déploiement de CCM s'attache à rester parfaitement indépendant de toute infrastructure d'exécution.
- Réciproquement, les ressources des grilles de calcul et les intergiciels qui permettent d'y accéder sont le plus souvent à usage générique, non dédiés à un type particulier d'applications.

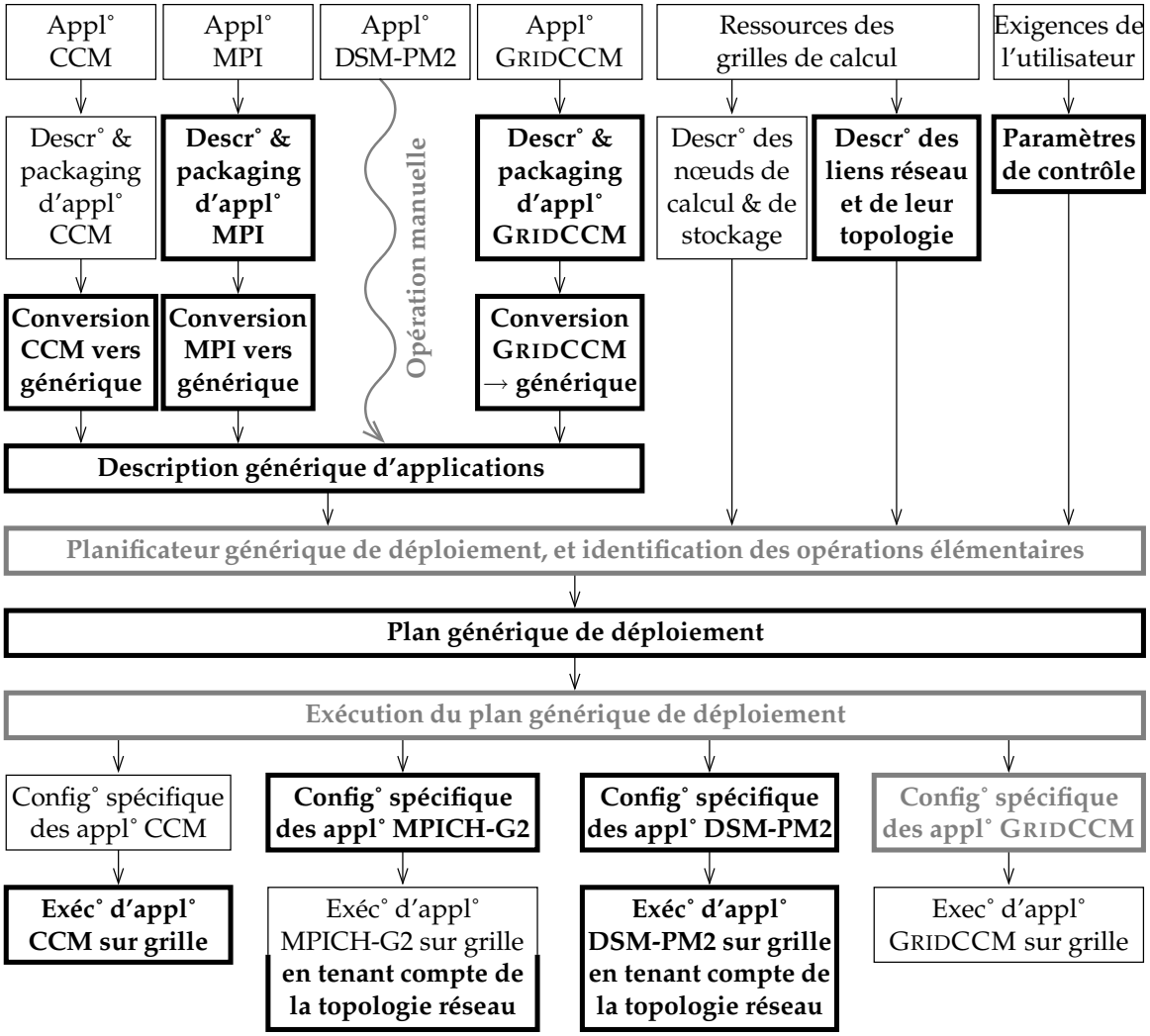


FIG. 12.2 – Déploiement automatique d'applications : nos contributions sont indiquées en **gras** ; les éléments grisés peuvent être encore largement approfondis.

Ainsi, le monde des applications et celui des grilles de calcul sont deux mondes orthogonaux (*cf.* figure 12.1) : les modèles des applications auxquelles nous nous intéressons n'imposent pas d'infrastructure d'exécution, et les grilles de calcul ne sont pas restreintes à un seul type d'application. Nos travaux ont construit un pont entre ces deux mondes pour les relier, en facilitant le déploiement d'applications sur des grilles de calcul.

Architecture de déploiement automatique. Après avoir *identifié les étapes du processus* de déploiement d'applications sur des grilles de calcul, nous avons proposé une architecture de déploiement *automatique*. Cette architecture prend la forme d'un outil de déploiement, intégré mais flexible. La figure 12.2 retrace cette architecture pour différents types d'applications, et résume nos contributions (indiquées en **gras**) telles que nous les détaillons dans la suite.

Au cœur de cette architecture se trouve le *planificateur de déploiement*. Il accepte en entrée la description de l'application à déployer, la description des ressources disponibles, ainsi que des

paramètres de contrôle, qui permettent à l'utilisateur de maîtriser le processus automatique de déploiement. Le planificateur est responsable de découvrir et sélectionner automatiquement les ressources nécessaires à l'exécution de l'application, placer les constituants de l'application, et choisir les implémentations de l'application qui devront être installées sur chaque ressource sélectionnée. En sortie, le planificateur de déploiement produit un *plan de déploiement*. L'exécution de ce plan consiste à transférer les fichiers (exécutables, données, dépendances, *etc.*) vers les ressources d'exécution, à lancer les programmes, et à configurer l'application.

Description de la topologie réseau des grilles. Les grilles de calcul mettent à notre disposition des ressources de calcul, de stockage, et de communication. L'état de l'art sait très bien décrire les nœuds de calcul et de stockage, mais pas les réseaux de communication entre ces nœuds. Nous avons proposé un modèle et un formalisme de *description de topologies réseau*, qui permettent de décrire des grilles complexes, mais réalistes. Ce modèle de description permet de représenter les pare-feux, les réseaux haute performance non IP, les liens asymétriques, les sous-réseaux en adresses IP privées, les passerelles de traduction d'adresses, *etc.* Il présente l'information de manière *compacte* (pour le passage à l'échelle) et *synthétique* : la description de la topologie réseau est de haut niveau d'abstraction, et donc facile à exploiter par le planificateur pour placer les constituants des applications.

Description spécifique et packaging d'applications MPI et GRIDCCM. Les applications à base de composants CCM possèdent déjà leur modèle de description. Pour pouvoir déployer automatiquement des applications MPI et GRIDCCM, nous avons défini des formalismes spécifiques de description pour ces applications. Ces descripteurs spécifiques contiennent toutes les informations utiles à l'outil de déploiement pour lancer les applications sans que l'utilisateur n'ait à intervenir pour compléter les informations. De plus, les descriptions spécifiques d'applications sont *indépendantes de toute infrastructure d'exécution* : cette propriété permet de déployer ces applications dans différents environnements sans avoir à modifier leurs descriptions. Pour la commodité de leur déploiement, les applications MPI et GRIDCCM peuvent être packagées de la même manière que les applications CCM.

Description générique d'applications. Un planificateur de déploiement est complexe à concevoir. Nous voulons donc *minimiser le nombre d'implémentations de planificateurs* pour chaque algorithme de planification. Ainsi, nous avons introduit la notion de *description générique d'applications* : chaque description spécifique d'application est traduite en description générique par un *simple convertisseur*. Cette dernière est donnée en entrée au planificateur, qui peut alors planifier le déploiement de n'importe quel type d'application. Notre modèle de description générique d'applications (GADe) est *indépendant de tout type d'application* : c'est une description de bas niveau, qui s'exprime dans des termes proches des infrastructures d'exécution, et qui décrit *explicitement* chaque entité de l'application à lancer.

Configuration des applications. La phase d'exécution du plan de déploiement se subdivise en deux étapes :

1. transfert des fichiers et lancement des processus sur les ressources ;
2. configuration de l'application.

La première étape peut en partie être réalisée indépendamment du type de l'application en cours de déploiement. Cependant, la deuxième étape est entièrement dépendante du type d'application. Notre architecture de déploiement vise à *factoriser* le plus possible les opérations de déploiement grâce à un mécanisme de *plugin* spécifique au type d'application à déployer. Ainsi, lorsque l'outil de déploiement doit effectuer une opération qui dépend du type d'application (générer une commande à lancer, configurer un composant, définir des variables d'environnement, *etc.*), il fait appel au plugin de l'application.

Adaptation des applications à la hiérarchie des performances réseau. Grâce au mécanisme de *configuration spécifique* d'application par le biais des plugins, les applications peuvent avoir accès à la connaissance de leur environnement d'exécution. En particulier, des fichiers de configuration ou des variables d'environnement peuvent décrire la topologie réseau sous-jacente de l'application. Ainsi, la bibliothèque MPICH-G2 et le système de MVP DSM-PM2 peuvent *s'adapter à la hiérarchie des performances de communication* afin de diminuer les temps d'exécution des applications.

Au delà des grilles de calcul. Ces travaux ont été motivés par les grilles de calcul. La variété et la complexité des topologies réseau des grilles nous ont conduits à définir un modèle de description des réseaux de communication. La diversité et la complexité des applications que les grilles peuvent héberger nous ont incités à définir une architecture d'automatisation du déploiement. Cependant, ces résultats ne sont pas spécifiques aux grilles de calcul : ils sont également applicables dans des environnements *plus simples*, tels que des réseaux locaux, des fédérations de clusters, *etc.*

Mise en œuvre et évaluation de performances

Nos résultats ont été mis en œuvre dans un outil de déploiement automatique baptisé ADAGE. Cet outil a permis de valider la simplicité, du point de vue de l'utilisateur, pour lancer une application packagée sur une grille de calcul. Des applications complexes telles que CCM, GRIDCCM, MPICH1-P4, MPICH-G2, JXTA peuvent maintenant se déployer en tapant simplement :

```
deploy -appl http://www.applications.fr/linux/fft/fast.zip
       -res ldap://mds.grid.org/mds-vo-name=local,o=grid
       -ctrl_params my_control_params.xml
```

où la description de l'application et la description des ressources sont *indépendantes*.

Enfin, les mesures de performance sur l'implémentation hiérarchique d'un système de MVP tel que DSM-PM2 renforce l'idée qu'il est nécessaire d'adapter les applications à l'hétérogénéité des ressources des grilles de calcul, afin de pouvoir exploiter toute leur puissance. Cependant, cette adaptation doit être aussi transparente que possible pour le développeur, en étant par exemple enfouie au sein des bibliothèques de programmation.

12.2 Perspectives

Ces travaux laissent la porte ouverte à divers prolongements. À court terme, il s'agit d'étendre les descriptions spécifiques et génériques des applications, d'enrichir la description des ressources, de classifier les paramètres de contrôle, ou encore d'étendre ADAGE pour qu'il gère l'intégralité du cycle de vie des applications. À moyen terme, nous pouvons imaginer de repenser la phase d'exécution du plan de déploiement, de supporter les applications dynamiques, de tolérer les défaillances, ou bien d'approfondir la planification.

12.2.1 Extensions envisageables à court terme

Descriptions spécifiques d'applications. Les formalismes de description spécifique d'applications sont extensibles, notamment pour préciser les *versions* des fichiers de données en entrée, des fichiers binaires (exécutables et DLL), des dépendances vis-à-vis de bibliothèques ou supports exécutifs (JVM JAVA, *etc.*), ainsi que pour les versions des systèmes d'exploitation et les architectures matérielles pour lesquels les fichiers binaires sont compilés. En général, plus les programmes compilés sont optimisés, et plus la spécification des versions doit être fine. En particulier, le pouvoir d'expression des versions doit permettre de tenir compte de la notion de compatibilité ascendante. Cette direction d'étude pourrait s'inspirer de la description des paquets Debian et des travaux sur le déploiement de *logiciels* (*cf.* section 4.1.1.1, page 53).

Tous les fichiers sont référencés par leur *localisation précise*, sous la forme d'une URL, qui détermine la localisation et le nom physique du fichier. Qu'il s'agisse de packages d'applications, de descripteurs d'applications, ou de fichiers de données en entrée, il se peut qu'ils soient répliqués sur les ressources disponibles. Il serait alors intéressant de référencer ces fichiers par des *noms logiques*, et un système de type pair-à-pair serait responsable de localiser une copie du fichier (la plus proche par exemple) et la rapatrier de manière transparente, éventuellement en parallèle depuis plusieurs sources.

En ce qui concerne la description spécifique d'applications MPI, les contraintes de partitionnement et la spécification des topologies virtuelles sont uniquement d'ordre *qualitatif*. Il pourrait être intéressant de pouvoir exprimer des contraintes d'ordre *quantitatif*, en spécifiant par exemple les rapports de débit et/ou de latence entre les processus d'un groupe et ceux localisés dans des groupes différents.

Description générique d'applications. La description spécifique des applications MPI permet d'exprimer des contraintes (sur la partitionnement et les topologies virtuelles) avec différents degrés de liberté quant au nombre de partitions de processus MPI ou au niveau de partitionnement (sur les clusters d'un site, sur les sites d'un pays, *etc.*). Pour simplifier le convertisseur de description spécifique d'applications MPI, et afin de laisser le planificateur faire ces choix, il serait intéressant de pouvoir *exprimer ces contraintes avec les mêmes degrés de liberté* dans GADe.

Il en est de même pour les contraintes sur les cardinalités des processus MPI. Par exemple, le formalisme de description spécifique d'application MPI permet d'exprimer que « au moins 90% des processus de l'application doivent être des processus esclaves ». Il serait intéressant de pouvoir traduire cette même contrainte dans la description générique d'applications, en définissant par exemple un *langage de contraintes*.

Description des ressources. Pour la description des ressources, les mêmes questions se posent que pour la description des applications, quant à la spécification des versions des ressources (systèmes d'exploitation, bibliothèques, *etc.*).

Les grilles de calcul possèdent des ressources de stockage : il pourrait donc être intéressant d'étendre la description des ressources avec les *méthodes de transfert de fichiers* supportées par les nœuds de stockage, de la même manière que les nœuds de calcul décrivent leurs méthodes de soumission de tâches.

Les description des ressources ne rend compte actuellement que des ressources matérielles et logicielles de très bas niveau (ordinateurs et leurs systèmes d'exploitation, réseaux de communication). Il sera de plus en plus intéressant de pouvoir intégrer la *description des services* dans la description des ressources disponibles, tels que des web services ou des daemons GridSolve par exemple. De même que le daemon SSH est un service qui permet de se connecter à une machine pour y ouvrir une session interactive, les daemons spécialisés devront pouvoir être décrits puisqu'ils permettent de lancer des tâches de calcul par exemple.

Formalisation des paramètres de contrôle. Dans ce document, nous avons mêlé tous les paramètres de contrôle dans un unique descripteur. Cependant, il apparaît que ces paramètres de contrôle interviennent soit au niveau des convertisseurs de descriptions spécifiques vers générique, soit pour la découverte des ressources, soit au niveau du planificateur, ou bien encore lors de l'exécution du plan de déploiement. Il pourrait alors être utile de *classer les paramètres de contrôle en différentes catégories*, afin de rationaliser cette notion.

Il pourrait aussi être intéressant de définir une *gradation entre les exigences de l'utilisateur* par le biais des paramètres de contrôle. En effet, ces exigences peuvent se révéler contradictoires. Par exemple, si les ressources disponibles sont un cluster de machines lentes avec un réseau rapide et un cluster de machines rapides avec un réseau lent, alors l'exigence d'un utilisateur qui voudrait exécuter son application avec les machines les plus puissantes et le réseau le plus rapide comporte une contradiction. Dans ce cas, il peut être intéressant d'attribuer un ordre d'importance aux paramètres de contrôle (plus ou moins souhaitables ou impératifs).

Gestion de l'intégralité du cycle de vie. Notre outil de déploiement ADAGE ne gère pas tout le cycle de vie d'une application : son utilité s'arrête après le lancement et la configuration de l'application. En particulier, ADAGE ne sait pas encore annuler l'exécution d'une application, même si les coordonnées de ses processus sont disponibles dans le rapport de déploiement. Il en est de même pour la désinstallation des fichiers mis en place avant exécution sur les différentes ressources. Avant de se lancer dans ce travail d'implémentation, il est utile d'étudier la possibilité de tuer un processus par une méthode différente de celle qui a été employée pour le lancer, si jamais la ressource offre plusieurs méthodes de soumission de tâches. C'est nécessaire par exemple lorsqu'une méthode de soumission de tâches devient défaillante.

Extension à d'autres technologies de programmation. Pour mettre plus encore à l'épreuve la généricité de GADe et la pertinence d'avoir un unique planificateur de déploiement pour différents types d'applications, il serait intéressant d'intégrer d'autres technologies de programmation dans ADAGE, telles que ProActive, CCA, PVM, SALOME[1], OPENMP, les EJB, Ju-

[1] SALOME : une plate-forme d'intégration pour des applications de simulation numérique, à base de composants logiciels (`http://www.salome-platform.org/`).

lia/Fractal, *etc.*

12.2.2 Prolongements envisageables à plus long terme

Décentralisation de la phase d'exécution du plan de déploiement. Dans ce document, nous avons donné une vision centralisée de l'exécution du plan de déploiement. L'outil de déploiement est utilisé sur un unique client depuis lequel le plan de déploiement est exécuté : tous les ordres individuels de transferts de fichiers et de lancements de tâches à distance sont donnés depuis ce client. Cependant, pour des raisons de passage à l'échelle, il pourrait être intéressant de *répartir les opérations d'exécution du plan de déploiement*, de façon à ce que les ordres de transferts de fichiers et de soumissions de tâches soient issus d'un ensemble de machines qui hébergent un « service permanent de déploiement ».

Planification du déploiement. Comme l'indique la section 5.2.3.1 (page 85), nous faisons la distinction entre les paramètres de contrôle de haut niveau et ceux de bas niveau, directement traduisibles en choix pour le planificateur par exemple. Il serait intéressant d'étudier comment *les paramètres de haut niveau peuvent être traduits en paramètres de bas niveau*, en collaboration avec le planificateur. Nous avons l'intuition que cette traduction nécessite de connaître le comportement des différentes implémentations de l'application en fonction des ressources, les performances des nœuds de calcul, les schémas de communication et les ratios calcul-communication des applications, les temps d'exécution souhaités par l'utilisateur, *etc.*

Pour que l'outil de déploiement automatique puisse faire des choix au moins aussi bons qu'un utilisateur qui lancerait manuellement son application, le planificateur doit être suffisamment sophistiqué. Il serait par exemple intéressant de mêler les résultats des travaux sur les algorithmes de détermination du nombre de processus pour une application, avec ceux de partitionnement et ceux de projection des topologies virtuelles sur des infrastructures d'exécution (*cf.* section 7.1.4, page 131). De plus, le planificateur devrait être capable de tenir compte des besoins de communication des applications, de la topologie réseau des ressources, et être en mesure de sélectionner les protocoles de communication lorsque plusieurs sont disponibles.

Enfin, la question de savoir si le planificateur doit décider absolument tout pour ne laisser aucune marge de décision à l'exécuteur du plan de déploiement est encore ouverte. Par exemple, le planificateur pourrait laisser le choix entre plusieurs méthodes de soumission de tâches, qui pourraient être tentées au moment de l'exécution du plan de déploiement. Ce serait utile si l'une des méthodes devait échouer, afin de ne pas faire avorter tout le processus de déploiement, et devoir tout recommencer, y compris la planification, en interdisant la méthode de soumission inopérante par le biais des paramètres de contrôle.

Dynamicité, adaptation et re-déploiement. Pour aborder le thème du déploiement automatique d'applications sur des grilles de calcul, nous nous sommes restreints, dans un premier temps, aux applications statiques. Cependant, les *applications dynamiques* prennent de plus en plus d'importance : connexion dynamique de programmes pour visualiser l'avancement d'un calcul, lancement de nouveaux processus avec MPI-2, exécution de workflow où les tâches sont lancées les unes à la suite des autres, en pipeline, en fonction des besoins, *etc.* Les applications dynamiques méritent également que l'automatisation de leur déploiement soit étudiée.

Pour ce faire, diverses questions attendent une réponse : la dynamicité peut-elle être traitée comme du *re-déploiement* ? Ou bien est-il utile d'avoir un daemon ou un service de déploiement permanent ?

Intuitivement, nous considérons le re-déploiement comme du déploiement avec des contraintes supplémentaires. Ces contraintes peuvent spécifier qu'une partie de l'application est déjà en cours d'exécution, et donner les coordonnées des processus déjà lancés. Elles pourraient s'exprimer par le biais des paramètres de contrôle.

Les mêmes questions se posent pour l'*adaptation des applications*, en réaction aux changements dans leur environnement d'exécution (charge des machines, congestion réseau, défaillances, *etc.*) : re-déploiement sous contraintes ou bien service permanent de déploiement ?

Tolérance aux défaillances. Pour tolérer les défaillances, l'application doit pouvoir s'adapter, et éventuellement se re-déployer, comme évoqué dans le paragraphe précédent.

En plus des défaillances lors de l'exécution de l'application, il peut être intéressant de tenir compte des défaillances ou des erreurs qui surviennent durant le processus de déploiement, notamment lors de la phase d'exécution de plan de déploiement. Dans ce cas, l'outil de déploiement peut être amené à « défaire » les opérations de déploiement effectuées (*roll-back*). Cette fonctionnalité est également utile si l'état des ressources change entre la phase de planification et celle d'exécution du plan de déploiement : faut-il alors rendre atomique la planification et l'exécution du plan, par une réservation transactionnelle des ressources ?

Bibliographie

Les premières références bibliographiques correspondent à nos propres publications. Elles sont listées à la page 4.

[13] David ABRAMSON, Rok SOSIC, Jonathan GIDDY et B. HALL. Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations. *In 4th IEEE Symposium on High Performance Distributed Computing (HPDC)*, pages 112–121. Washington, DC, USA, août 1995.

[14] Benjamin A. ALLAN, Robert C. ARMSTRONG, Alicia P. WOLFE, Jaideep RAY, David E. BERNHOLDT et James A. KOHL. The CCA Core Specification In a Distributed Memory SPMD Framework. *Concurrency and Computation : Practice and Experience*, volume 14, n° 5, pages 323–345, 2002.

[15] Gabrielle ALLEN, Kelly DAVIS, Konstantinos N. DOLKAS, Nikolaos D. DOULAMIS, Tom GOODALE, Thilo KIELMANN, André MERZKY, Jarek NABRZYSKI, Juliusz PUKACKI, Thomas RADKE, Michael RUSSELL, Ed SEIDEL, John SHALF et Ian TAYLOR. Enabling Applications on the Grid: a GridLab Overview. *International Journal of High Performance Computing Applications (JHPCA), special issue on Grid Computing : Infrastructure and Applications*, volume 17, n° 4, août 2003.

[16] Gabrielle ALLEN, Kelly DAVIS, Tom GOODALE, Andrei HUTANU, Hartmut KAISER, Thilo KIELMANN, Andre MERZKY, Rob van NIEUWPOORT, Alexander REINEFELD, Florian SCHINTKE, Thorsten SCHÜTT, Ed SEIDEL et Brygg ULLMER. The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid. *Proceedings of the IEEE*, volume 93, n° 3, mars 2005.

[17] Gabrielle ALLEN, Thomas DRAMLITSCH, Ian FOSTER, Nicholas T. KARONIS, Matei RIPEANU, Edward SEIDEL et Brian TOONEN. Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus. *In SuperComputing Conference*, pages 52–76. Denver, CO, USA, novembre 2001.

[18] Giovanni ALOISIO, Massimo CAFARO, Italo EPICOCO et Sandro FIORE. Analysis of the Globus Toolkit Grid Information Service. Rapport technique GridLab-10-D.1-0001-1.0, HPCC, University of Lecce, Italie, 2002.

[19] Giovanni ALOISIO, Massimo CAFARO, Italo EPICOCO, Daniele LEZZI, Maria MIRTO, Silvia MOCAVERO et Serena PATI. First GridLabMDS Release. Rapport technique GridLab-10-D.3-0001-1.0, HPCC, University of Lecce, Italy, 2002.

[20] Cristiana AMZA, Alan L. COX, Sandhya DWARKADAS, Pete KELEHER, Honghui LU, Ramakrishnan RAJAMONY, Weimin YU et Willy ZWAENEPOEL. TreadMarks: Shared Memory Computing on Networks of Workstations. *IEEE Computer*, volume 29, n° 2, pages 18–28, février 1996.

[21] Ali ANJOMSHOAA, Fred BRISARD, Michel DRESCHER, Donal FELLOWS, An LY, Stephen MCGOUGH, Darren PULSIPHER et Andreas SAVVA. Job Submission Description Language (JSDL) Specification. Draft specification, version 1.0, Global Grid Forum (GGF), juin 2005.

[22] Gabriel ANTONIU. *DSM-PM2 : une plate-forme portable pour l'implémentation de protocoles de cohérence multithreads pour systèmes à mémoire virtuellement partagée*. Thèse de doctorat, Laboratoire d'Informatique du Parallélisme (LIP), École Normale Supérieure de Lyon, France, novembre 2001.

[23] Gabriel ANTONIU, Luc BOUGÉ et Mathieu JAN. JuxMem: An Adpative Supportive Platform for Data Sharing on the Grid. *In ACM Workshop on Adaptive Grid Middleware (AGridM 2003)*, pages 49–59. New Orleans, Lousiane, USA, septembre 2003.

[24] Gabriel ANTONIU, Luc BOUGÉ, Mathieu JAN et Sébastien MONNET. Large-scale Deployment in P2P Experiments Using the JXTA Distributed Framework. *In Euro-Par Conference (Parallel Processing)*, volume 3149 de *LNCS*, pages 1038–1047. Springer-Verlag, Pise, Italie, août 2004.

[25] Luciana Bezerra ARANTES, Pierre SENS et Bertil FOLLIOT. An Effective Logical Cache for a Clustered LRC-Based DSM System. *Cluster Computing Journal*, volume 5, n° 1, pages 19–31, janvier 2002.

[26] Rob ARMSTRONG, Dennis GANNON, Al GEIST, Katarzyna KEAHEY, Scott KOHN, Lois MCINNES, Steve PARKER et Brent SMOLINSKI. Toward a Common Component Architecture for High-Performance Scientific Computing. *In 8th IEEE International Symposium on High Performance Distributed Computing (HPDC'99)*, pages 115–124. Redondo Beach, CA, USA, août 1999.

[27] Dorian ARNOLD, Sudesh AGRAWAL, Susan BLACKFORD, Jack DONGARRA, Christoph FABIANEK, Tomo HIROYASU, Eric MEEK, Michelle MILLER, Kiran SAGI, Keith SEYMOUR, Zhiao SHI et Sathish VADHIYAR. Users' Guide to NetSolve V2.0. Innovative computing deptartment technical report, University of Tennessee, Knoxville, TN, juin 2003.

[28] Olivier AUMAGE, Luc BOUGÉ, Jean-François MÉHAUT et Raymond NAMYST. Madeleine II: A Portable and Efficient Communication Library for High-Performance Cluster Computing. *Parallel Computing*, volume 28, n° 4, pages 607–626, 2002.

[29] Laurent BADUEL, Françoise BAUDE et Denis CAROMEL. Efficient, Flexible, and Typed Group Communications in Java. *In 2002 joint ACM-ISCOPE conference on Java Grande*, pages 28–36. ACM Press, Seattle, WA, USA, novembre 2002.

[30] Henri E. BAL, Aske PLAAT, Mirjam G. BAKKER, Peter DOZY et Rutger F. H. HOFMAN. Optimizing Parallel Applications for Wide-Area Clusters. *In 12th International Parallel Processing Symposium, 9th Symposium on Parallel and Distributed Processing (IPPS/SPDP)*, pages 784–790. IEEE Computer Society, Orlando, Florida, USA, avril 1998.

[31] Donald B. BATCHELOR. Integrated Simulation of Fusion Plasmas. *Physics Today*, pages 35–40, février 2005.

[32] Francoise BAUDE, Denis CAROMEL et Matthieu MOREL. From Distributed Objects to Hierarchical Grid Components. *In International Symposium on Distributed Objects and Applications (DOA)*, numéro 2888 de LNCS, pages 1226–1242. Springer Verlag, Catania, Italie, novembre 2003.

[33] Françoise BAUDE, Denis CAROMEL, Lionel MESTRE, Fabrice HUET et Julien VAYSSIÈRE. Interactive and Descriptor-based Deployment of Object-Oriented Grid Applications. *In 11th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 93–102. IEEE Computer Society, Edinburgh, Scotland, UK, juillet 2002.

[34] David BELL, Takashi KOJO, Patrick GOLDSACK, Steve LOUGHRAN, Dejan MILOJICIC, Stuart SCHAEFER, Junichi TATEMURA et Peter TOFT. Configuration Description, Deployment, and Lifecycle Management (CDDLM). Foundation document, Global Grid Forum (GGF), août 2005.

[35] Felipe BERTRAND et Randall BRAMLEY. DCA: A distributed CCA framework based on MPI. *In 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS)*, pages 80–89. Santa Fe, NM, USA, avril 2004.

[36] Joseph BESTER, Ian FOSTER, Carl KESSELMAN, Jean TEDESCO et Steven TUECKE. GASS: A Data Movement and Access Service for Wide Area Computing Systems. *In 6th Workshop on Input/Output in Parallel and Distributed Systems (IOPADS)*, pages 78–88. ACM Press, Atlanta, GA, mai 1999.

[37] Jim BLYTHE, Ewa DEELMAN, Yolanda GIL, Carl KESSELMAN, Amit AGARWAL, Gaurang MEHTA et Karan VAHI. The Role of Planning in Grid Computing. *In 13th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 153–163. AAAI Press, Trento, Italie, juin 2003.

[38] Nanette J. BODEN, Danny COHEN, Robert E. FELDERMAN, Alan E. KULAWIK, Charles L. SEITZ, Jakov N. SEIZOVIC et Wen-King SU. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, volume 15, nº 1, pages 29–36, février 1995.

[39] Tracy D. BRAUN, Howard Jay SIEGEL, Noah BECK, Ladislau L. BÖLÖNI, Muthucumaru MAHESWARAN, Albert I. REUTHER, James P. ROBERTSON, Mitchell D. THEYS, Bin YAO, Debra HENSGEN et Richard F. FREUND. A Comparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing Systems. *In 8th Heterogeneous Computing Workshop (HCW)*, pages 15–29. San Juan, Puerto Rico, avril 1999.

[40] Yuri BREITBART, Minos N. GAROFALAKIS, Cliff MARTIN, Rajeev RASTOGI, S. SESHADRI et Abraham SILBERSCHATZ. Topology Discovery in Heterogeneous IP Networks. *In IEEE INFOCOM'2000*, pages 265–274. Tel-Aviv, Israël, mars 2000.

[41] Frédéric BRICLET, Christophe CONTRERAS et Philippe MERLE. OpenCCM : une infrastructure à composants pour le déploiement d'applications à base de composants CORBA. *In 1ère Conférence Francophone sur le Déploiement et la (Re)Configuration de Logiciels (DECOR'2004)*, pages 101–112. Grenoble, France, octobre 2004.

[42] Matthias BRUNE, Alexander REINEFELD et Jörg VARNHOLT. A Resource Description Environment for Distributed Computing Systems. *In 8th IEEE International Symposium on High Performance Distributed Computing (HPDC'99)*, pages 279–286. Redondo Beach, CA, USA, août 1999.

[43] Éric BRUNETON. Fractal Packaging Specification, juin 2004. URL : `http://mail-archive.objectweb.org/architecture/2004-06/msg00004.html`. Draft version 0.1.

[44] Éric BRUNETON, Thierry COUPAYE et Jean-Bernard STÉFANI. The Fractal Component Model. Draft Specification version 2.0-3, The ObjectWeb Consortium, février 2004.

[45] Rajkumar BUYYA, David ABRAMSON et Jon GIDDY. Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid. *In 4th*

International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia), pages 283–289. IEEE Computer Society, Pékin, Chine, mai 2000.

[46] Kenneth L. CALVERT, Matthew B. DOAR et Ellen W. ZEGURA. Modeling Internet Topology. *IEEE Communications Magazine*, volume 35, n° 6, pages 160–163, juin 1997.

[47] Franck CAPPELLO, Samir DJILALI, Gilles FEDAK, Thomas HERAULT, Frédéric MAGNIETTE, Vincent NÉRI et Oleg LODYGENSKY. Computing on Large Scale Distributed Systems: XtremWeb Architecture, Programming Models, Security, Tests and Convergence with Grid. *Future Generation Computer Science (FGCS)*, volume 21, n° 3, pages 417–437, mars 2005.

[48] Eddy CARON et Holly DAIL. GoDIET: a tool for managing distributed hierarchies of DIET agents and servers. Research report RR-5520, INRIA, mars 2005.

[49] Eddy CARON, Frédéric DESPREZ, Frédéric LOMBARD, Jean-Marc NICOD, Martin QUINSON et Frédéric SUTER. A Scalable Approach to Network Enabled Servers. *In 8th International Euro-Par Conference*, édité par B. MONIEN et R. FELDMANN, volume 2400 de *LNCS*, pages 907–910. Springer Verlag, août 2002.

[50] Henri CASANOVA, Graziano OBERTELLI, Francine BERMAN et Rich WOLSKI. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. *In SuperComputing Conference (SC'2000)*, pages 60–78. Dallas, TX, USA, novembre 2000.

[51] CORBA Components. OMG Document formal/02-06-65, Object Management Group, juin 2002. Version 3.0.

[52] Humberto CERVANTES, Mikael DÉSERTOT et Didier DONSEZ. FROGi : déploiement de composants Fractal sur OSGi. *In 1ère Conférence Francophone sur le Déploiement et la (Re)Configuration de Logiciels (DECOR'2004)*, pages 147–158. Grenoble, France, octobre 2004.

[53] Common Object Request Broker Architecture: Core Specification. OMG Document formal/04-03-12, Object Management Group, mars 2004. Version 3.0.3.

[54] R. Les COTTRELL, Connie LOGG et I-Heng MEI. Experiences and Results from a New High Performance Network and Application Monitoring Toolkit. *In Passive and Active Measurement Workshop (PAM2003)*, pages 205–217. La Jolla, CA, USA, avril 2003.

[55] Karl CZAJKOWSKI, Steven FITZGERALD, Ian FOSTER et Carl KESSELMAN. Grid Information Services for Distributed Resource Sharing. *In 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC)*, pages 181–194. San Francisco, CA, août 2001.

[56] Karl CZAJKOWSKI, Ian FOSTER, Nick KARONIS, Carl KESSELMAN, Stuart MARTIN, Warren SMITH et Steven TUECKE. A Resource Management Architecture for Metacomputing Systems. *In IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1459 de *LNCS*, pages 62–82. 1998.

[57] Mathias DALHEIMER et Donal Fellows Soonwook HWANG. OGSA Resource Selection Services Working Group. Charter, GGF, août 2005.

[58] Ewa DEELMAN, James BLYTHE, Yolanda GIL, Carl KESSELMAN, Scott KORANDA, Albert LAZZARINI, Gaurang MEHTA, Maria Alessandra PAPA et Karan VAHI. Pegasus and the Pulsar Search: From Metadata to Execution on the Grid. *In 5th International Conference on Parallel Processing and Applied Mathematics (PPAM)*, volume 3019 de *LNCS*, pages 821–830. Czestochowa, Pologne, septembre 2003.

[59] Ewa DEELMAN, Gurmeet SINGH, Mei-Hui SU, James BLYTHE, Yolanda GIL, Carl KESSELMAN, Gaurang MEHTA, Karan VAHI, G. Bruce BERRIMAN, John GOOD, Anastasia LAITY, Joseph C. JACOB et Daniel S. KATZ. Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems, 2005. Soumis à *Scientific Programming*.

[60] Thomas A. DEFANTI, Ian FOSTER, Michael E. PAPKA, Rick STEVENS et Tim KUHFUSS. Overview of the I-Way: Wide-Area Visual Supercomputing. *The International Journal of Supercomputer Applications*, volume 10, n° 2, pages 123–130, 1996.

[61] Mathijs den BURGER, Thilo KIELMANN et Henri E. BAL. TopoMon: A Monitoring Tool for Grid Network Topology. *In International Conference on Computational Science, part 2 (ICCS2002)*, numéro 2330 de LNCS, pages 558–567. Springer-Verlag, Amsterdam, Pays-Bas, avril 2002.

[62] Alexandre DENIS. *Contribution à la conception d'une plate-forme haute performance d'intégration d'exécutifs communicants pour la programmation des grilles de calcul*. Thèse de doctorat, Université de Rennes 1, IRISA, Rennes, France, décembre 2003.

[63] Alexandre DENIS, Christian PÉREZ, et Thierry PRIOL. Network Communications in Grid Computing: At a Crossroads Between Parallel and Distributed Worlds. *In 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, page 95a. IEEE Computer Society, Santa Fe, NM, USA, avril 2004.

[64] Alexandre DENIS, Christian PÉREZ et Thierry PRIOL. Towards High Performance CORBA and MPI Middlewares for Grid Computing. *In 2nd International Workshop on Grid Computing*, édité par Craig A. LEE, numéro 2242 de LNCS, pages 14–25. Springer-Verlag, Denver, CO, USA, novembre 2001. En conjonction avec SuperComputing 2001 (SC'01).

[65] Alexandre DENIS, Christian PÉREZ, Thierry PRIOL et André RIBES. Padico: A Component-Based Software Infrastructure for Grid Computing. *In 17th International Parallel and Distributed Processing Symposium (IPDPS 2003)*. IEEE Computer Society, Nice, France, avril 2003.

[66] Peter DINDA et Beth PLALE. A Unified Relational Approach to Grid Information Services. Informational Draft GWD-GIS-012-1, Global Grid Forum (GGF), février 2001.

[67] Matthew B. DOAR. A Better Model for Generating Test Networks. *In IEEE Global Telecommunications Conference (Globecom'96)*, pages 86–93. Londres, UK, novembre 1996.

[68] Suchuan DONG, George Em KARNIADAKIS et Nicholas T. KARONIS. Cross-Site Computations on the TeraGrid. *Computing in Science & Engineering (CiSE)*, volume 7, n° 5, pages 14–23, septembre 2005.

[69] Pierre-François DUTOT, Lionel EYRAUD, Grégory MOUNIÉ et Denis TRYSTRAM. Scheduling on Large Scale Distributed Platforms: From Models to Implementations. *International Journal of Foundations of Computer Science*, volume 16, n° 2, pages 217–237, 2005.

[70] Linda G. DeMichiel *et al*. Enterprise JavaBeans™ Specification. Final Release Version 2.1, Sun Microsystems, août 2003. URL : `http://Java.Sun.com/products/ejb/docs.html`.

[71] John FEO, David C. CANN et R. R. OLDEHOEFT. A Report on the SISAL Language Project. *Journal of Parallel and Distributed Computing (JPDC)*, volume 10, n° 4, pages 349–366, décembre 1990.

[72] Areski FLISSI et Philippe MERLE. Vers un environnement multi personnalités pour la configuration et le déploiement d'applications à base de composants logiciels. *In 1ère Conférence Francophone sur le Déploiement et la (Re)Configuration de Logiciels (DECOR'2004)*, pages 3–14. Grenoble, France, octobre 2004.

[73] Areski FLISSI et Philippe MERLE. Une démarche dirigée par les modèles pour construire les machines de déploiement des intergiciels à composants. *In Langages et Modèles à Objets (LMO 2005)*, pages 79–94. Hermès Sciences, Berne, Suisse, mars 2005.

[74] Ian FOSTER. What is the Grid? A Three Point Checklist. *GRIDToday*, volume 1, n° 6, juillet 2002.

[75] Ian FOSTER, Jonathan GEISLER, Bill NICKLESS, Warren SMITH et Steven TUECKE. Software Infrastructure for the I-Way High Performance Distributed Computing Experiment. *In 5th IEEE Symposium on High Performance Distributed Computing (HPDC)*, pages 562–571. Syracuse, NY, août 1996.

[76] Ian FOSTER et Carl KESSELMAN. The Globus Project: a Status Report. *In 7th Heterogeneous Computing Workshop, held in conjunction with IPPS/SPDP'98*, pages 4–18. Orlando, FL, mars 1998.

[77] Ian FOSTER et Carl KESSELMAN (éditeurs). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA, janvier 1998.

[78] Ian FOSTER et Carl KESSELMAN (éditeurs). *The Grid: Blueprint for a New Computing Infrastructure*, chapitre The Globus Toolkit, pages 259–278. Morgan Kaufmann, San Francisco, CA, janvier 1998.

[79] Ian FOSTER, Carl KESSELMAN, Jeffrey M. NICK et Steven TUECKE. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Draft, Global Grid Forum (GGF), juin 2002.

[80] Ian FOSTER, Carl KESSELMAN et Steven TUECKE. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *The International Journal of Supercomputer Applications*, volume 15, n° 3, pages 200–222, 2001.

[81] Ian T. FOSTER, Carl KESSELMAN, Gene TSUDIK et Steven TUECKE. A Security Architecture for Computational Grids. *In 5th ACM Conference on Computer and Communications Security*, pages 83–92. ACM Press, New York, NY, San Francisco, CA, 1998.

[82] Ian T. FOSTER, Robert OLSON et Steven TUECKE. Productive Parallel Programming: The PCN Approach. *Journal of Scientific Programming*, volume 1, n° 1, pages 51–66, 1992.

[83] James FREY, Todd TANNENBAUM, Miron LIVNY, Ian FOSTER et Steven TUECKE. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *In 10th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 55–63. San Francisco, CA, août 2001.

[84] Nathalie FURMENTO, , William LEE, Steven NEWHOUSE et John DARLINGTON. Test and Deployment of ICENI, an Integrated Grid Middleware on the UK e-Science Grid. *In UK e-Science All Hands Meeting*, pages 192–195. Nottingham, UK, septembre 2003.

[85] Nathalie FURMENTO, Anthony MAYER, Stephen MCGOUGH, Steven NEWHOUSE et John DARLINGTON. A Component Framework for HPC Applications. *In 7th International Euro-Par Conference*, volume 2150 de *LNCS*, pages 540–548. Manchester, UK, août 2001.

[86] Nathalie FURMENTO, Anthony MAYER, Stephen MCGOUGH, Steven NEWHOUSE, Tony FIELD et John DARLINGTON. Optimisation of Component-based Applications within

a Grid Environment. *In 2001 ACM/IEEE conference on Supercomputing*, page 30. ACM Press, New York, NY, USA, Denver, CO, USA, novembre 2001.

[87] Nathalie FURMENTO, Anthony MAYER, Stephen MCGOUGH, Steven NEWHOUSE, Tony FIELD et John DARLINGTON. ICENI: Optimisation of Component Applications within a Grid Environment. *Journal of Parallel Computing*, volume 28, n° 12, pages 1753–1772, 2002.

[88] Al GEIST, Adam BEGUELIN, Jack DONGARRA, Weicheng JIANG, Robert MANCHEK et Vaidy SUNDERAM. *PVM: Parallel Virtual Machine — A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, novembre 1994.

[89] Patrick GOLDSACK, Julio GUIJARRO, Antonio LAIN, Guillaume MECHENEAU, Paul MURRAY et Peter TOFT. SmartFrog: Configuration and Automatic Ignition of Distributed Applications. *In HP OpenView University Association conference (HP OVUA)*. Genève, Suisse, juillet 2003.

[90] Andrew S. GRIMSHAW et William A. WULF. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, volume 40, n° 1, janvier 1997.

[91] William GROPP, Ewing LUSK et Anthony SKJELLUM. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, MA, USA, 2ème édition, novembre 1999.

[92] William GROSSO. *Java RMI*. O'Reilly & Associates, 1ère édition, octobre 2001.

[93] Richard S. HALL, Dennis HEIMBIGNER et Alexander L. WOLF. A Cooperative Approach to Support Software Deployment Using the Software Dock. *In International Conference on Software Engineering (ICSE)*, pages 174–183. Los Angeles, CA, USA, mai 1999.

[94] Dennis HEIMBIGNER, Richard S. HALL et Alexander L. WOLF. A Framework for Analyzing Configurations of Deployable Software Systems. *In 5th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 32–42. Las Vegas, NV, USA, octobre 1999.

[95] High Performance FORTRAN Language Specification. High Performance Fortran Forum, janvier 1997. Version 2.0.

[96] Adriana IAMNITCHI et Ian FOSTER. On Fully Decentralized Resource Discovery in Grid Environments. *In 2nd International Workshop on Grid Computing*, édité par Craig A. LEE, volume 2242 de *LNCS*, pages 51–62. Springer-Verlag, Denver, CO, USA, novembre 2001.

[97] Liviu IFTODE. *Home-based Shared Virtual Memory*. Thèse de Ph.D., Princeton University, NJ, USA, juin 1998.

[98] Anca-Andreea IVAN, Josh HARMAN, Michael ALLEN et Vijay KARAMCHETI. Partitionable Services: A Framework for Seamlessly Adapting Distributed Applications to Heterogeneous Environments. *In 11th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 103–112. IEEE Computer Society, Edimbourg, Écosse, UK, juin 2002.

[99] JAVA Web Start Overview. White paper, Sun Microsystems, mai 2005.

[100] Nicholas KARONIS, Bronis de SUPINSKI, Ian FOSTER, William GROPP, Ewing LUSK et John BRESNAHAN. Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance. *In 14th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 377–384. Cancun, Mexique, mai 2000.

[101] Nicholas T. KARONIS, Brian TOONEN et Ian FOSTER. MPICH-G2: a Grid-enabled implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing (JPDC)*, volume 63, n° 5, pages 551–563, 2003.

[102] Peter J. KELEHER, Alan L. COX, Sandhya DWARKADAS et Willy ZWAENEPOEL. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. *In Winter 1994 USENIX Conference*, pages 115–132. San Francisco, CA, USA, janvier 1994.

[103] Axel KELLER et Alexander REINEFELD. Anatomy of a Resource Management System for HPC Clusters. Rapport technique 00-38, Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Allemagne, novembre 2000.

[104] Rainer KELLER, Bettina KRAMMER, Matthias S. MUELLER, Michael M. RESCH et Edgar GABRIEL. MPI Development Tools and Applications for the Grid. *In Workshop on Grid Applications and Programming Tools*. Seattle, WA, USA, juin 2003. Présenté à GGF8.

[105] Tatiana KICHKAYLO, Anca-Andreea IVAN et Vijay KARAMCHETI. Constrained Component Deployment in Wide-Area Networks Using AI Planning Techniques. *In 17th International Parallel and Distributed Processing Symposium (IPDPS)*, page 3. Nice, France, avril 2003.

[106] Tatiana KICHKAYLO et Vijay KARAMCHETI. Optimal Resource-Aware Deployment Planning for Component-based Distributed Applications. *In 13th International Symposium on High Performance Distributed Computing (HPDC)*, pages 150–159. Honolulu, HI, USA, juin 2004.

[107] Thilo KIELMANN, Henri E. BAL, Sergei GORLATCH, Kees VERSTOEP et Rutger F. H. HOFMAN. Network Performance-aware Collective Communication for Clustered Wide Area Systems. *Journal of Parallel Computing*, volume 27, n° 11, pages 1431–1456, 2001.

[108] Thilo KIELMANN, Rutger F. H. HOFMAN, Henri E. BAL, Aske PLAAT et Raoul BHOEDJANG. MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems. *In 1999 ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP'99)*, pages 131–140. Atlanta, GA, USA, mai 1999.

[109] Thilo KIELMANN, Rutger F. H. HOFMAN, Henri E. BAL, Aske PLAAT et Raoul A. F. BHOEDJANG. MPI's Reduction Operations in Clustered Wide Area Systems. *In Message-Passing Interface Developer's and User's Conference (MPIDC'99)*, pages 43–52. Atlanta, GA, USA, mars 1999.

[110] Sriram KRISHNAN et Dennis GANNON. XCAT3: A Framework for CCA Components as OGSA Services. *In 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS)*, pages 90–97. Santa Fe, NM, USA, avril 2004.

[111] Krzysztof KUROWSKI, Bogdan LUDWICZAK, Ariel OLEKSIAK, Tomasz PIONTEK et Juliusz PUKACKI. GRMS User Guide version 1.9.6. Rapport technique, GridLab, Poznan Supercomputing and Networking Center, mars 2005.

[112] Zakaria LAHJOMRI et Thierry PRIOL. KOAN: A Shared Virtual Memory for the iPSC/2 Hypercube. *In 2nd Joint International Conference on Vector and Parallel Processing (CONPAR/VAPP'92)*, édité par Luc BOUGÉ, Michel COSNARD, Yves ROBERT et Denis TRYSTRAM, volume 634 de *LNCS*, pages 441–452. Lyon, France, septembre 1992.

[113] Vincent LESTIDEAU et Noureddine BELKHATIR. Providing Highly automated and generic means for software deployment Process. *In 9th European Workshop on Software Process Technology (EWSPT)*, volume 2786 de *LNCS*, pages 128–142. Springer-Verlag, Helsinki, Finland, septembre 2003.

[114] Kai LI. *Shared Virtual Memory on Loosely Coupled Multiprocessors*. Thèse de Ph.D., Yale University, New Haven, CT, USA, octobre 1986.

[115] Kai LI et Paul HUDAK. Memory Coherence in Shared Virtual Memory Systems. *ACM Transactions on Computer Systems*, volume 7, n° 4, pages 321–359, novembre 1989.

[116] Greg LINDAHL. The Case for an MPI ABI. Présentation, 2005. URL : `http://www.openib.org/docs/oib_wkshp_022005/mpi-abi-pathscale-lindahl.pdf`.

[117] Bruce LOWEKAMP, Brian TIERNEY, Les COTTRELL, Richard HUGHES-JONES, Thilo KIELMANN et Martin SWANY. A Hierarchy of Network Performance Characteristics for Grid Applications and Services. Proposed recommendation, Network Measurement Working Group (NMWG), Global Grid Forum (GGF), janvier 2004.

[118] Bruce B. LOWEKAMP, Nancy MILLER, Dean SUTHERLAND, Thomas GROSS, Peter STEENKISTE et Jaspal SUBHLOK. A Resource Query Interface for Network-Aware Applications. *Journal of Cluster Computing*, volume 2, n° 2, pages 139–151, 1999.

[119] Bruce B. LOWEKAMP, Brian TIERNEY, Les COTTRELL, Richard HUGHES-JONES, Thilo KIELMANN et Martin SWANY. Enabling Network Measurement Portability Through a Hierarchy of Characteristics. *In 4th International Workshop on Grid Computing (Grid2003)*, pages 68–75. Phoenix, AZ, USA, novembre 2003.

[120] Dong LU et Peter A. DINDA. Synthesizing Realistic Computational Grids. *In SuperComputing 2003 (SC'03)*, page 16. Phoenix, AZ, USA, novembre 2003.

[121] Gnanamanikam MAHINTHAKUMAR, Forrest M. HOFFMAN, William W. HARGROVE et Nicholas T. KARONIS. Multivariate Geographic Clustering in a Metacomputing Environment Using Globus. *In SuperComputing Conference*, pages 5–16. Portland, OR, USA, novembre 1999.

[122] Yves MAHÉO, Frédéric GUIDEC et Luc COURTRAI. Middleware Support for the Deployment of Resource-Aware Parallel JAVA Components on Heterogeneous Distributed Platforms. *In 30th EUROMICRO Conference*, pages 144–151. IEEE Computer Society, Rennes, France, septembre 2004.

[123] Loris MARCHAL, Yang YANG, Henri CASANOVA et Yves ROBERT. A Realistic Network/Application Model for Scheduling Divisible Loads on Large-Scale Platforms. *In 19th International Parallel and Distributed Processing Symposium (IPDPS)*, page 48. IEEE Computer Society, Denver, CO, USA, avril 2005.

[124] Raphaël MARVIE, Philippe MERLE, Jean-Marc GEIB et Mathieu VADET. OpenCCM : une plate-forme ouverte pour composants CORBA. *In 2ème conférence Française sur les Systèmes d'Exploitation (CFSE)*, page 112. Paris, France, avril 2001.

[125] Sangman MOH, Chansu YU, Hee Yong YOUN, Ben LEE et Dongsoo HAN. Mapping Strategies for Switch-Based Cluster Systems of Irregular Topology. *In 8th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 733–740. IEEE Computer Society, Kyongju City, Corée, juin 2001.

[126] Myricom. *GM: A Message-Passing System for Myrinet Networks*. URL : `http://www.myri.com/scs/GM-2/doc/html/`. Manuel de référence.

[127] Myricom. *Myrinet Express (MX): A High-Performance, Low-Level, Message-Passing Interface for Myrinet*. URL : `http://www.myri.com/scs/#documentation`.

[128] Raymond NAMYST. *PM2 : un environnement pour une conception portable et une exécution efficace des applications parallèles irrégulières*. Thèse de doctorat, LIFL, Université de Lille 1, janvier 1997.

[129] Katia OBRACZKA et Grig GHEORGHIU. The Performance of a Service for Network-Aware Applications. *In 2nd ACM Symposium on Parallel and Distributed Tools (SPDT'98)*, pages 81–91. Welches, OR, USA, août 1998.

[130] Deployment and Configuration of Component-based Distributed Applications Specification. Draft Adopted Specification ptc/03-07-02, OMG, juin 2003.

[131] Juan M. ORDUÑA, Federico SILLA et José DUATO. A New Task Mapping Technique for Communication-Aware Scheduling Strategies. *In 30th International Conference on Parallel Processing Workshops (ICPPW)*, pages 349–354. IEEE Computer Society, Valencia, Espagne, septembre 2001.

[132] Pradeep PADALA et Joesph N. WILSON. GridOS: Operating System Services for Grid Architectures. *In 10th International Conference On High Performance Computing (HiPC)*, pages 353–362. Hyderabad, Inde, décembre 2003.

[133] Karl PAULS et Richard S. HALL. Eureka: A Resource Discovery Service for Component Deployment. *In 2nd International Working Conference on Component Deployment (CD 2004)*, édité par Wolfgang EMMERICH et Alexander L. WOLF, volume 3083 de *LNCS*, pages 159–174. Springer-Verlag, Édimbourg, Écosse, UK, mai 2004.

[134] Laura PEARLMAN, Von WELCH, Ian FOSTER, Carl KESSELMAN et Steven TUECKE. A Community Authorization Service for Group Collaboration. *In 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 50–59. Monterey, CA, juin 2002.

[135] Aske PLAAT, Henri E. BAL, Rutger F.H. HOFMAN et Thilo KIELMANN. Sensitivity of Parallel Applications to Large Differences in Bandwidth and Latency in Two-Layer Interconnects. *Future Generation Computer Systems*, volume 17, n° 6, pages 769–782, avril 2001.

[136] 1003.4d8 POSIX System Application Program Interface: Threads Extensions [C language]. Rapport technique, IEEE Standards Department, 1994.

[137] Loïc PRYLLI et Bernard TOURANCHEAU. BIP: a new protocol designed for high performance networking on Myrinet. *In 12th International Parallel Processing Symposium (IPPS) ; 9th Symposium on Parallel and Distributed Processing (SPDP)*, volume 1388, pages 472–485. Springer-Verlag, Orlando, FL, USA, mars 1998.

[138] Loïc PRYLLI et Bernard TOURANCHEAU. BIP: a New Protocol Designed for High Performance Networking on Myrinet. *In 1st Workshop on Personal Computer based Networks Of Workstations (PC-NOW'98)*, LNCS, pages 472–485. Springer-Verlag, Orlando, FL, USA, avril 1998.

[139] Christian PÉREZ, Thierry PRIOL et André RIBES. A Parallel CORBA Component Model for Numerical Code Coupling. *International Journal of High Performance Computing Applications (IJHPCA)*, volume 17, n° 4, pages 417–429, 2003.

[140] Rajesh RAMAN, Miron LIVNY et Marvin SOLOMON. Matchmaking: Distributed Resource Management for High Throughput Computing. *In 7th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 140–146. Chicago, IL, USA, juillet 1998.

[141] Michel RAYNAL. *Algorithms for Mutual Exclusion*. MIT Press, Cambridge, MA, août 1986.

[142] André RIBES. *Contribution à la conception d'un modèle de programmation parallèle et distribué et sa mise en œuvre au sein de plates-formes orientées objet et composant*. Thèse de doctorat, Université de Rennes 1, IRISA, Rennes, France, décembre 2004.

[143] Matei RIPEANU et Ian FOSTER. A Decentralized, Adaptive Replica Location Mechanism. *In 11th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 24–32. Edimbourg, Écosse, juillet 2002.

[144] Jose L. RUIZ, Juan C. DUEÑAS, Fernando USERO et Cristina DÍAZ. Deployment in dynamic environments. *In 1ère Conférence Francophone sur le Déploiement et la (Re)Configuration de Logiciels (DECOR'2004)*, pages 85–96. Grenoble, France, octobre 2004.

[145] Jennifer M. SCHOPF et Bill NITZBERG. Grids: The top ten questions. *Scientific Programming*, volume 10, n° 2, pages 103–111, août 2002.

[146] IEEE Standard for Scalable Coherent Interface (SCI). Rapport technique, IEEE Standard 1596, août 1993.

[147] Gary SHAO, Fran BERMAN et Rich WOLSKI. Using Effective Network Views to Promote Distributed Application Performance. *In 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, volume 5. Las Vegas, NV, USA, juin 1999.

[148] Marc SNIR, Steve OTTO, Steven HUSS-LEDERMAN, David WALKER et Jack DONGARRA. *MPI – The Complete Reference, The MPI-1 Core*, volume 1. MIT Press, Cambridge, MA, USA, 2ème édition, septembre 1998.

[149] Pyda SRISURESH et Matt HOLDREGE. IP Network Address Translator (NAT) Terminology and Considerations. Informational RFC 2663, IETF Network Working Group, août 1999.

[150] Martin SWANY et Rich WOLSKI. Representing Dynamic Performance Information in Grid Environments with the Network Weather Service. *In 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'02)*, pages 48–56. Berlin, Allemagne, mai 2002.

[151] Clemens SZYPERSKI. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1ère édition, 1998.

[152] Yoshio TANAKA, Hiroshi TAKEMIYA, Hidemoto NAKADA et Satoshi SEKIGUCHI. Design, implementation and performance evaluation of GridRPC programming middleware for a large-scale computational Grid. *In 5th IEEE/ACM International Workshop on Grid Computing*, pages 298–305. Pittsburgh, PA, USA, novembre 2004.

[153] Douglas THAIN, Todd TANNENBAUM et Miron LIVNY. Condor and the Grid. *In Grid Computing : Making the Global Infrastructure a Reality*, édité par Fran BERMAN, Geoffrey FOX et Tony HEY. John Wiley & Sons Inc., décembre 2002.

[154] Rajeev THAKUR et William D. GROPP. Improving the Performance of Collective Operations in MPICH. *In 10th European PVM/MPI Users' Group Meeting (PVM/MPI)*, volume 2840 de *LNCS*, pages 257–267. Venise, Italie, septembre 2003.

[155] THE GLOBUS PROJECT. GridFTP: Universal Data Transfer for the Grid. Whitepaper, University of Chicago, septembre 2000.

[156] THE GLOBUS PROJECT. GridFTP update. Rapport technique, Argonne National Laboratory, janvier 2002.

[157] Brian TIERNEY, Ruth AYDT, Dan GUNTER, Warren SMITH, Martin SWANY, Valerie TAYLOR et Rich WOLSKI. A Grid Monitoring Architecture. Informational draft, Global Grid Forum (GGF), août 2002.

[158] Arthur van HOFF, Hadi PARTOVI et Tom THAI. The Open Software Description Format (OSD). Submitted note, W3C, août 1997. URL : `http://www.w3.org/TR/NOTE-OSD.html`.

[159] Manuela VELOSO, Jaime CARBONELL, Alicia PÉREZ, Daniel BORRAJO, Eugene FINK et Jim BLYTHE. Integrating Planning and Learning: The PRODIGY Architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, volume 7, n° 1, pages 81–120, 1995.

[160] Werner VOGELS. Web Services Are Not Distributed Objects. *IEEE Internet Computing*, volume 7, n° 6, 2003.

[161] Rich WOLSKI, Neil SPRING et Jim HAYES. The Network Weather Service: a Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, volume 15, n° 5-6, pages 757–768, octobre 1999.

[162] Steven Cameron WOO, Moriyoshi OHARA, Evan TORRIE, Jaswinder Pal SINGH et Anoop GUPTA. The SPLASH-2 Programs: Characterization and Methodological Considerations. *In 22nd International Symposium on Computer Architecture*, pages 24–36. Santa Margherita Ligure, Italie, juin 1995.

[163] Min-You WU et Wei SHU. An Efficient Distributed Token-Based Mutual Exclusion Algorithm with Central Coordinator. *Journal of Parallel and Distributed Computing (JPDC)*, volume 62, n° 10, pages 1602–1613, octobre 2002.

[164] Laurie YOUNG, Stephen MCGOUGH, Steven NEWHOUSE et John DARLINGTON. Scheduling Architecture and Algorithms within the ICENI Grid Middleware. *In UK e-Science All Hands Meeting*, pages 5–12. Nottingham, UK, septembre 2003. ISBN : 1-904425-11-9.

[165] Keming ZHANG, Kostadin DAMEVSKI, Venkatanand VENKATACHALAPATHY et Steven G. PARKER. SCIRun2: A CCA Framework for High Performance Computing. *In 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS)*, pages 72–79. Santa Fe, NM, USA, avril 2004.

[166] Yuanyuan ZHOU, Liviu IFTODE et Kai LI. Performance Evaluation of Two Home-Based Lazy Release Consistency Protocols for Shared Memory Virtual Memory Systems. *In 2nd Symposium on Operating Systems Design and Implementation (OSDI'96)*, pages 75–88. Seattle, WA, USA, octobre 1996.

[167] *Assembly and Deployment Toolkit*. URL : `http://www.fpx.de/MicoCCM/Toolkit/`.

[168] BOINC. URL : `http://boinc.berkeley.edu/`.

[169] Exemples d'applications en C++. URL : `http://www.research.att.com/~bs/applications.html`.

[170] CDDLM Working Group. URL : `https://forge.gridforum.org/projects/cddlm-wg`.

[171] Centre Européen pour la Recherche Nucleaire. URL : `http://www.CERN.ch/`.

[172] Condor. URL : `http://www.cs.wisc.edu/condor/`.

[173] Cray XT3. URL : `http://www.Cray.com/products/xt3/index.html`.

[174] DataTAG. URL : `http://DataTAG.web.cern.ch/datatag/`.

[175] Microsoft Distributed Component Object Model. URL : `http://www.MicroSoft.com/com/`.

[176] DIET. URL : `http://graal.ens-lyon.fr/DIET/`.

[177] DOE Science Grid. URL : `http://DOEScienceGrid.org/`.

[178] EGEE: Enabling Grids for E-sciencE in Europe. URL : `http://EGEE-ei.web.CERN.ch/egee-ei/`.

[179] Elagi. URL : `http://grail.sdsc.edu/projects/elagi/`.

[180] FFTW. URL : `http://www.fftw.org/`.

[181] Fractal. URL : `http://Fractal.Objectweb.org/`.

[182] FROGi: Fractal over OSGi. URL : `http://www-adele.imag.fr/frogi/`.

[183] GGF: Global Grid Forum. URL : `http://www.GGF.org/`.

[184] The Globus Alliance. URL : `http://www.Globus.org/`.

[185] Grid'5000. URL : `http://www.Grid5000.org/`.

[186] ENV. URL : `http://grail.sdsc.edu/projects/env/GridML.html`.

[187] GridOS. URL : `http://www.eecs.umich.edu/~ppadala/research/gridos/`.

[188] NetSolve / GridSolve. URL : `http://icl.cs.utk.edu/netsolve/`.

[189] GridWay. URL : `http://www.gridway.org/`.

[190] GRid Interoperability Project. URL : `http://www.Grid-Interoperability.org/`.

[191] GriPhyN: Grid Physics Network. URL : `http://www.GriPhyN.org/`.

[192] High Performance FORTRAN Forum. URL : `http://dacnet.rice.edu/Depts/CRPC/HPFF/`.

[193] Le projet HydroGrid. URL : `http://www-rocq.inria.fr/~kern/HydroGrid/HydroGrid.html`.

[194] IBM Grid Computing. URL : `http://www-1.IBM.com/grid/`.

[195] ICENI. URL : `http://www.lesc.ic.ac.uk/iceni/`.

[196] InfiniBand. URL : `http://www.InfiniBandTA.org/ibta/`.

[197] InstallShield. URL : `http://www.InstallShield.com/`.

[198] ITER. URL : `http://www.iter.org/`.

[199] JXTA Distributed Framework (JDF). URL : `http://jdf.jxta.org/`.

[200] Julia Tutorial. URL : `http://Fractal.Objectweb.org/tutorials/julia/index.html`.

[201] JXTA. URL : `http://www.jxta.org/`.

[202] Legion: Worldwide Virtual Computer. URL : `http://www.cs.virginia.edu/~legion/`.

[203] Load Sharing Facility. URL : `http://www.Platform.com/products/LSF/`.

[204] Globus MDS. URL : `http://www.Globus.org/toolkit/mds/`.

[205] MPICH-G2. URL : `http://www.Globus.org/mpi/`.

[206] MPI Forum (standards, archives, *etc.*). URL : `http://www.MPI-Forum.org/`.

[207] Myricom, technologie réseau Myrinet. URL : `http://www.myri.com/`.

[208] Nimrod. URL : `http://www.csse.monash.edu.au/nimrod/`.

[209] Ninf-G. URL : `http://ninf.apgrid.org/`.

[210] OASIS. URL : `http://www.oasis-open.org/`.

[211] ObjectWeb: Open Source Middleware. URL : http://www.Objectweb.org/.

[212] OpenCCM. URL : http://OpenCCM.objectweb.org/.

[213] OSCAR, une implémentation de OSGi. URL : http://OSCAR.objectweb.org/.

[214] Open Services Gateway Initiative (OSGi). URL : http://www.OSGi.org/.

[215] PACX-MPI. URL : http://www.hlrs.de/organization/pds/projects/pacx-mpi/.

[216] Padico. URL : http://www.irisa.fr/paris/Padico/.

[217] Portable Batch System (OpenPBS). URL : http://www.OpenPBS.org/.

[218] PETSc. URL : http://www.MCS.ANL.gov/petsc/.

[219] ProActive. URL : http://www-sop.inria.fr/oasis/ProActive/.

[220] Quadrics. URL : http://www.Quadrics.com/.

[221] RENATER. URL : http://www.RENATER.fr/.

[222] RFT: Reliable File Transfer Service. URL : http://www.Globus.org/toolkit/docs/3.2/rft/index.html.

[223] RLS: Replica Location Service. URL : http://www.Globus.org/toolkit/docs/3.2/rls/key/index.html.

[224] RSL Specification 1.0. URL : http://www-fp.Globus.org/gram/rsl_spec1.html.

[225] Resource Selection Services Working Group (GGF). URL : https://forge.gridforum.org/projects/ogsa-rss-wg/.

[226] ScaLAPACK. URL : http://www.netlib.org/scalapack/scalapack_home.html.

[227] OASIS Solution Deployment Descriptor (SDD). URL : http://xml.coverpages.org/OASIS-SDD-CFP.html.

[228] Sun Grid Engine. URL : http://www.Sun.com/software/gridware/.

[229] SGI Altix 3000. URL : http://www.SGI.com/servers/altix/.

[230] SGI Origin 3000. URL : http://www.SGI.com/products/servers/origin/3000/.

[231] SmartFrog. URL : http://www.smartfrog.org/.

[232] Sun Grid. URL : http://www.Sun.com/service/sungrid/overview.jsp.

[233] TeraGrid. URL : http://www.TeraGrid.org/.

[234] UNICORE Forum. URL : http://www.UNICORE.org/.

[235] XtremWeb. URL : http://www.xtremweb.net/.

Sébastien LACOUR

CONTRIBUTION À L'AUTOMATISATION DU DÉPLOIEMENT D'APPLICATIONS SUR DES GRILLES DE CALCUL

Mots-clefs : grilles informatiques, parallélisme, systèmes distribués, déploiement d'applications, topologie réseau, MPI, CCM, GRIDCCM, ADAGE.

Résumé

Le déploiement d'applications de calcul scientifique sur des grilles informatiques est un problème difficile, du fait de la complexité des applications et de l'hétérogénéité des grilles. Dans le but de masquer ces difficultés, cette thèse propose une architecture qui automatise le processus de déploiement d'applications comprenant plusieurs programmes qui communiquent entre eux sur des grilles de calcul. Nous proposons également un modèle de description de la topologie réseau complexe des grilles, ainsi que des modèles de description spécifique d'applications parallèles et d'applications mixtes (à la fois parallèles et distribuées). Nous introduisons la notion de description générique d'applications, qui permet à un unique planificateur de déploiement de sélectionner les ressources d'exécution et de placer les constituants des applications. Ces contributions sont validées par la mise en œuvre d'un outil de déploiement automatique.