Intergiciel pour l'exécution efficace et fiable d'applications distribuées dans des grilles dynamiques de très grande taille

Emmanuel Jeanvoine

N° d'ordre: 3595

THÈSE

Présentée devant

l'Université de Rennes 1

pour obtenir

le grade de : DOCTEUR DE L'UNIVERSITÉ DE RENNES 1
Mention INFORMATIQUE

par

Emmanuel JEANVOINE

Équipe d'accueil : INRIA - Équipe-Projet PARIS
École Doctorale : Matisse
Composante universitaire : IFSIC

Titre de la thèse :

Intergiciel pour l'exécution efficace et fiable d'applications distribuées dans des grilles dynamiques de très grande taille

soutenue le 27 novembre 2007 devant la commission d'examen

M.	Thierry	PRIOL	Président
MM.	Frédéric	DESPREZ	Rapporteurs
	Pierre	SENS	
MM.	Jean-Yves	BERTHOU	Examinateurs
	Domenico	LAFORENZA	
	Daniel	LEPRINCE	
Mme.	Christine	MORIN	Directrice de thèse

Remerciements

Voilà ! Trois ans ... c'est long, mais ça passe si vite ! La thèse est un effort de fond qui ne peut pas être réalisé par une seule personne, par conséquent je vais tenter de remercier ici ceux qui m'ont aidé au cours de ces trois dernières années de travail. D'avance, merci à ceux que j'aurais oublié.

J'adresse tout d'abord mes plus vifs remerciement aux membres du jury. Merci à Frédéric Desprez et à Pierre Sens d'avoir accepté aussi rapidement de rapporter cette thèse en dépit d'un emploi du temps très chargé. Merci à Thierry Priol de m'avoir fait l'honneur de présider ce jury et merci à Domenico Laforenza de s'être intéressé à mes travaux en dépit de la langue française utilisée dans ce manuscrit.

Je tiens à remercier très sincèrement Christine Morin pour son encadrement pendant cette thèse, pour m'avoir fait confiance dès le début et pour avoir su me rassurer dans mes moments de doute. Malgré de nombreuses responsabilités, Christine m'a accordé le temps qu'il fallait lorsque cela était nécessaire.

Merci à Daniel Leprince pour m'avoir accompagné dans le quotidien de mes séjours à EDF et pour avoir été toujours disponible pour répondre à mon avalanche de questions techniques comme administratives. Merci également pour l'autonomie qu'il m'a laissée durant cette thèse.

Merci à Jean-Yves Berthou pour m'avoir fait confiance au démarrage de cette thèse lorsqu'il m'a accueilli dans l'équipe I26 de SINETICS. La vision *long-terme* des domaines du calcul à hautes performances et de la simulation numérique qu'il a apportée à cette thèse a été cruciale pour cerner les problématiques et enjeux industriels.

Merci aux membres des équipes I2A et I2D du département SINETICS à EDF pour leur accueil lors de mon arrivée et pour leur bonne humeur permanente. Un merci tout particulier à Christian C, Éric F., Hugues P., Samuel K., Vincent G., Vincent L., Yvan D.-M.. Merci bien entendu au marseillais André pour sa coopération, son soutien et sa bonne humeur.

Je remercie également les membres (ou ex-membres) de l'équipe-projet PARIS de l'INRIA. Plus particulièrement, merci à : Adrien L., Christian P., David M., Étienne R., Jean P., Loïc C., Matthieu F., Najib A., Pascal G., Pascal M., Renaud L., Yvon J.. Merci à Louis pour sa grande sympathie et pour le temps qu'il a consacré à répondre à toutes mes questions lorsque j'ai commencé à travailler sur Vigne avec lui. Merci à Hinde pour son soutien : je te souhaite plein de bonnes choses pour l'avenir. Merci à Boris pour nos mémorables discussions et pour ta détermination qui pourrait servir de modèle à beaucoup : je te souhaite également beaucoup de bien et d'accomplir tout ce qui te plaira. Jérôme et Thomas, il a été très agréable et intéressant de travailler avec vous : bon vent à vous deux.

Merci à ma famille d'avoir cru en moi et de m'avoir donné les moyens de suivre la voie qui me plaisait. Merci infiniment à Aude pour avoir sacrifié d'innombrables heures à relire ce manuscrit composé de choses qu'elle ne comprenait pas, et évidemment, merci pour tout le reste aussi. Finalement, merci à tous ceux qui ont croisé mon chemin à Rennes, à Besançon ou encore à Pontarlier, d'une façon ou d'une autre vous avez forcément influencé ce travail.

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Introduction

Pour améliorer notre environnement quotidien, la communauté scientifique modélise les éléments qui nous entourent afin de mieux les comprendre. Ces modélisations sont fondées sur de nombreux calculs dont la résolution est complexe et fastidieuse. Pour gagner du temps, pour augmenter la précision des modélisations ou pour éviter les erreurs, les moyens informatiques permettent d'automatiser la résolution des calculs. Ainsi, la communauté scientifique a toujours été un élément moteur dans le développement des moyens de calcul informatiques.

Dans l'évolution des moyens informatiques dédiés à la résolution de problèmes scientifiques, plusieurs générations se sont succédées. En trente ans, l'architecture des calculateurs a considérablement évolué en affichant des cadences processeur qui sont passées de quelques kilo-hertz à plusieurs giga-hertz et des quantités de mémoire qui sont passées de quelques kilo-octets à plusieurs giga-octets. Cela a permis de traiter un nombre d'opérations toujours croissant dans un temps identique et de modéliser des problèmes de plus en plus complexes. Cette évolution fut caractérisée par la loi de Moore qui prévoit que tous les 18 mois le nombre de transistors dans un processeur d'entrée de gamme est doublé. Cette loi a été globalement vérifiée de 1975 à 2004, mais un ralentissement est actuellement constaté suite à des difficultés liées à la dissipation thermique. Conjointement à ces améliorations, l'architecture interne des processeurs a également été améliorée en utilisant plusieurs étages d'exécution (*pipelines*) pour paralléliser le traitement des instructions et en utilisant des caches toujours plus performants qui permettent de limiter les accès à la mémoire vive.

La science avançant à grands pas et les problèmes à résoudre étant toujours plus complexes, la puissance d'un seul calculateur n'était plus suffisante. L'ère du parallélisme est donc apparue. L'idée est de diviser un problème complexe en plusieurs problèmes simples afin de pouvoir les résoudre sur plusieurs processeurs simultanément. Pour cela, des machines multi-processeurs et des super-calculateurs ont été construits en incorporant un grand nombre de processeurs et une large quantité de mémoire. Plusieurs déclinaisons de ces super-calculateurs ont vu le jour, chacune ayant ses spécificités par rapport au schéma de traitement des instructions et par rapport au schéma d'accès des différents processeurs à la mémoire. Dans un même temps l'évolution des réseaux locaux a permis d'assembler des calculateurs individuels pour construire des grappes, moins performantes que les machines parallèles mais beaucoup moins coûteuses et ainsi, très largement adoptées.

Les technologies des réseaux à longue distance évoluant également, les infrastructures tout d'abord constituées de quelques calculateurs interconnectés pour un usage militaire ont cédé leur place à l'Internet, une toile tissée sur les quatre coins de la planète qui interconnecte des centaines de millions d'ordinateurs. Afin de supporter les activités liées à l'Internet grand pu-

blic, de solides fondations ont été mises en place dans le cœur du réseau, notamment grâce aux technologies fondées sur l'utilisation de la fibre optique. Les technologies liées ou dérivées d'Internet permettant ainsi d'interconnecter de façon très efficace un grand nombre de calculateurs, la communauté scientifique y a vu une nouvelle chance pour étendre ses moyens de calcul. C'est ainsi que les grilles sont apparues. Dorénavant, plusieurs centres de calcul composés de super-calculateurs, de grappes et de simples calculateurs peuvent être interconnectés pour résoudre un même problème.

Grâce à la puissance de calcul fournie par les grilles, la communauté scientifique peut envisager d'innombrables applications. En particulier, la simulation numérique permet de modéliser des phénomènes qui ne pourraient pas être vérifiés en situation réelle pour des raisons de coûts, de sécurité ou encore de temps. La simulation numérique est utilisée pour de nombreuses applications comme la thermo-dynamique des fluides, la mécanique des structures, l'électromagnétisme, la génétique ou encore l'hydrologie.

Si les grilles constituent une solution de choix pour résoudre de nouveaux problèmes de plus en plus complexes, leur utilisation reste néanmoins beaucoup plus complexe qu'une station de travail, qu'une grappe ou qu'un super-calculateur. Les ressources peuvent à tout moment être retirées de la grille de façon programmée, dans le cas d'une opération de maintenance par exemple, ou de façon non programmée, dans le cas d'une défaillance par exemple. Cette volatilité, accompagnée du fait que la grille soit composée d'un grand nombre de ressources, potentiellement plusieurs milliers, complexifie l'utilisation puisque les utilisateurs ne peuvent pas connaître simplement à un moment donné quelles sont les ressources disponibles. De plus, les applications pour grille possèdent potentiellement une architecture plus complexe que celles destinées à des plates-formes d'une échelle plus réduite. Typiquement, la conception de ces applications peut être fondée sur l'utilisation de plusieurs grains de parallélisme. Ainsi, certaines parties d'une application peuvent être optimisées pour une exécution sur des machines possédant un réseau rapide et d'autres parties peuvent s'affranchir de cette contrainte. Avec la grande taille d'une grille, un utilisateur pourra difficilement exécuter au mieux ce type d'application puisque connaître la localisation et les spécificités de chaque machine de la grille s'avère fastidieux et difficile.

En parallèle de l'avancée des caractéristiques matérielles des moyens de calcul, l'offre logicielle pour les exploiter a considérablement évolué. Depuis une vingtaine d'années, de nombreux travaux de recherche ont été dédiés à l'élaboration de services permettant d'utiliser des machines distribuées. Condor [116] est, par exemple, un projet pionnier dans le domaine dont l'objectif est de récupérer les cycles processeur inutilisés sur les stations de travail d'une institution. Par la suite, des projets comme Legion [66], Globus [84] ou encore UNICORE [147] ont vu le jour en visant l'exploitation de grilles de plus grande taille, non limitées à une institution.

Depuis le début des années 2000, plusieurs tentatives de standardisation des services de gestion des moyens de calcul d'une grille sont apparues. Alors que l'*Open Grid Forum* [23] (fusion du *Global Grid Forum* et de l'*Enterprise Grid Alliance*), tente de fournir les briques qui permettent de construire un système de gestion des ressources pouvant inter-opérer avec d'autres systèmes, d'autres projets comme Gridbus [61] ou GridLab [11] ont pour objectif de définir l'architecture toute entière d'un système pour grille.

Contexte

Cette thèse s'inscrit dans le cadre d'une collaboration CIFRE entre le département SINETICS d'EDF Recherche& Développement, et l'équipe-projet PARIS de l'INRIA.

EDF, et plus particulièrement sa direction Recherche et Développement, mène de nombreuses études liées à la production et à la distribution d'énergie dans un contexte de sécurité et d'efficacité. Typiquement, la construction d'une centrale nucléaire requiert une rigueur telle que la création de chaque constituant doit être finement étudiée. Pour cela, de nombreuses simulations numériques sont conduites dans les domaines de la neutronique, de la mécanique des fluides et des structures ou encore dans le domaine de la thermo-dynamique. Les moyens de calcul à EDF ont considérablement augmenté depuis les dix dernières années si bien qu'en novembre 2006, EDF possédait un calculateur parallèle à la 61^{e} place du TOP500 [34].

Les grappes et les calculateurs, dont le nombre croît constamment, interconnectés à travers les différents sites d'EDF constituent désormais une infrastructure matérielle de grille. La gestion des ressources du point de vue des utilisateurs et des administrateurs devient de plus en plus complexe si bien que de nombreuses heures sont consacrées uniquement aux procédures d'utilisation des ressources. Cette complexité d'accès peut avoir un effet dissuasif sur les utilisateurs et peut engendrer une sous-exploitation des ressources.

À l'INRIA, l'équipe-projet PARIS s'intéresse à la programmation de systèmes parallèles et distribués de grande taille. Depuis 2002, il s'intéresse en particulier à la conception d'un système intégré pour les grilles de très grande taille. L'objectif est de permettre l'exécution fiable et efficace d'une large gamme d'applications tout en offrant une interface simple aux utilisateurs. Dans le cadre de sa thèse [143], Louis Rilling a conçu l'architecture du système Vigne qui offre un ensemble de services système fiables permettant d'offrir aux utilisateurs une vision de type *système à image unique* (SIU) des ressources d'une grille de très grande taille. Il a plus spécifiquement étudié un service de communication fondé sur l'utilisation d'un réseau logique structuré qui permet de garantir la vue SIU et un service de partage de données volatiles pour programmer des applications utilisant le concept de *mémoire partagée*.

Le contexte général de notre travail se situe dans la continuité des travaux menés dans l'équipe-projet PARIS, en accord avec la volonté d'EDF R&D de rationaliser l'accès à ses moyens de calcul pour l'exécution d'applications de simulation numérique.

Objectifs de la thèse

Dans cette thèse, nous nous intéressons à la problématique de la gestion des ressources dans les grilles de calcul de grande taille. En terme d'échelle, nous visons des grilles constituées de plusieurs milliers ou plusieurs dizaines de milliers de nœuds qui peuvent être répartis dans plusieurs pays.

Notre objectif principal est de concevoir un système conciliant trois propriétés caractérisant l'exécution d'applications sur les ressources d'une grille de calcul de grande taille qui sont : la simplicité, la fiabilité et l'efficacité. De plus, ce système doit être capable d'exécuter une large gamme d'applications pouvant avoir une architecture distribuée et pouvant être patrimoniales.

Comme nous l'avons énoncé précédemment, l'utilisation des ressources d'une grille est

beaucoup plus complexe que l'utilisation d'un calculateur ou d'une grappe, principalement à cause de leur volatilité, de leur hétérogénéité et de la grande échelle. Ces caractéristiques doivent être occultées aux utilisateurs car ces derniers, souvent non informaticiens, ne sont pas forcément des experts face à l'utilisation de ressources distribuées et volatiles. De plus, les tâches d'administration consacrées à l'agrégation des ressources de différents domaines d'administration doivent être simplifiées au maximum pour inciter les fournisseurs de ressources à se joindre à la grille et pour les soulager d'opérations fastidieuses.

Les défaillances de ressources sont le quotidien du monde de la grille. Or de nombreux projets de l'état de l'art laissent à l'utilisateur le soin de vérifier que l'exécution de son application sur la grille n'a pas été interrompue par une défaillance. Pour affranchir l'utilisateur de ces opérations, le système doit garantir que les applications atteindront le terme de leur exécution en dépit des défaillances, si tant est que les codes applicatifs soient exempts d'erreur. Le système doit lui-même être capable de survivre à la défaillance de n'importe quel constituant de la grille afin d'assurer une continuité dans le service pour l'utilisation des nœuds en état de marche.

La gestion d'un grand nombre de nœuds dispersés sur plusieurs sites géographiques peut s'avérer coûteuse en ressources. Beaucoup d'approches de l'état de l'art reposent sur une architecture centralisée où le passage à l'échelle est compromis par le risque de goulet d'étranglement induit par de nombreux accès à des serveurs centraux. Ce risque peut être limité en dimensionnant en conséquences les ressources de calcul et la connectivité des éléments centraux mais dans ce cas le coût devient prohibitif. Un système fédérant les ressources d'une grille doit permettre de s'affranchir d'une solution centrale qui serait coûteuse et qui compromettrait le passage à l'échelle. Un système sera capable de fédérer efficacement un grand nombre de ressources s'il n'impose pas de ressources de calcul ou de liens de communication dédiés.

Finalement, les applications de simulation numérique couramment utilisées dans l'industrie, et en particulier à EDF, possèdent des architectures très variées. Ainsi, le système doit offrir un support pour exécuter de façon efficace ces applications. Par exemple, les éléments applicatifs fortement communicants devront être placés par le système sur des nœuds possédant une bonne interconnexion entre-eux, et ceci sans que l'utilisateur n'ait à spécifier la localisation des nœuds à utiliser. Sachant que de nombreuses applications, dites patrimoniales, sont uniquement disponibles sous forme binaire, le système de grille ne doit pas imposer leur modification pour les exécuter efficacement sur une grille. Ici encore, les approches de l'état de l'art ne sont pas entièrement satisfaisantes puisque la modification des applications est nécessaire pour profiter d'un support efficace pour les exécutions, ce qui exclut l'exécution d'applications patrimoniales.

Contributions

Cette thèse présente la conception et la mise en œuvre de services système pour la gestion de ressources et l'exécution d'applications distribuées dans des grilles de très grande taille. Les contributions que nous avons apportées à l'état de l'art sont les suivantes.

Tout d'abord, nous avons proposé une architecture générale pour la gestion des ressources d'une grille. Cette architecture étend l'architecture du système Vigne proposée par Louis

Rilling et repose sur des réseaux logiques pair-à-pair. Cela confère à Vigne d'excellentes propriétés de passage à l'échelle et de tolérance aux défaillances. Cette architecture fournit également une vision de type *système à image unique* de l'ensemble des nœuds d'une grille en occultant leur distribution aux utilisateurs. C'est un facteur essentiel pour la simplification de l'utilisation d'une grille.

Dans l'architecture proposée, nous avons étudié plus particulièrement la conception de trois services.

Nous avons proposé un système d'information permettant de recenser les ressources présentes dans la grille afin de les utiliser pour l'exécution des applications des utilisateurs. Ce système d'information est fondé sur un réseau logique pair-à-pair non structuré et sur le concept de marches aléatoires optimisé pour le contexte d'utilisation des grilles de calcul. La recherche de ressources dans la grille, sans être exhaustive, permet de satisfaire au mieux les besoins d'une application.

Ensuite, nous avons proposé un service d'allocation de ressources permettant de répartir au mieux les applications soumises sur les ressources de la grille afin d'éviter les exécutions concurrentes sur un même nœud qui pénalisent le temps de complétion des applications. Nous avons également proposé un mécanisme permettant d'allouer simultanément plusieurs nœuds (co-allocation de ressources) pour l'exécution d'applications patrimoniales composées de tâches communicantes. Afin d'exécuter le plus efficacement possible ce dernier type d'application, notre service d'allocation de ressources permet d'allouer des nœuds ayant une interconnexion réseau efficace entre-eux pour minimiser le coût des communications entre les tâches.

Nous avons aussi proposé un service de gestion d'application permettant une exécution fiable en dépit des défaillances des nœuds. Ce service prend en charge le pilotage d'applications complexes dont nous décrivons la structure à l'aide de trois types de dépendances entre les tâches applicatives. Ces dépendances permettent de décrire l'architecture d'une large gamme d'applications distribuées comme celles fondées sur les modèles suivants : sac de tâches, maître/travailleur, couplage de code ou encore *workflow*.

Ces trois services ne reposent sur aucun point de centralisation. Ils contribuent ainsi à la construction d'un système fiable, auto-organisant, auto-réparant et adapté à la grande taille d'une grille. Par ailleurs, cette architecture distribuée est capable de s'adapter à tous les modèles de grilles comme les réseaux de stations de travail, les fédérations de grappes ou encore les grilles hétérogènes.

Nous avons réalisé un prototype des services que nous avons conçus. Celui-ci a été intégré au système Vigne. Ce prototype fonctionnel permet de soumettre simplement des applications qui seront exécutées de façon fiable sur les nœuds d'une grille exploités en interactif ou avec un gestionnaire de traitement par lots. Le déploiement du prototype ainsi que les opérations de maintenance du point de vue de l'administrateur sont largement simplifiés par rapport aux approches de l'état de l'art.

Nous avons également couplé le système Vigne à la plate-forme industrielle de simulation numérique SALOME [31] afin de faire bénéficier cette dernière des ressources d'une grille. Pour cela, nous avons proposé une extension du noyau de SALOME.

Nous avons évalué notre prototype par simulation et en conditions réelles d'exécution sur un grand nombre de nœuds de la plate-forme Grid'5000 [57]. Nous avons ainsi montré la

faisabilité des concepts avancés dans cette thèse et l'efficacité des services proposés.

Par rapport aux travaux de l'état de l'art, Vigne est comparable à des systèmes intégrés visant la gestion des ressources des grilles de très grande taille comme Vishwa [168] ou Zorilla [73]. Comme Globus [84], Vigne est suffisamment générique pour fédérer les ressources de n'importe quel type de grilles et pour exécuter n'importe quel type d'applications tout en offrant des fonctionnalités de plus haut niveau qui simplifient considérablement l'utilisation et l'administration d'une grille. En revanche, les problématiques liées à la sécurité dans les grilles comme cela est traité par Globus [84] ou UNICORE [147] sortent du cadre de nos travaux. Toutefois, l'intégration de mécanismes de sécurité ne remettrait pas en cause l'architecture globale de Vigne. Finalement, Vigne ne nécessite pas de modifier le système d'exploitation comme MOSIX [48] ou 9Grid [122].

Organisation du document

Ce document est organisé de la façon suivante. Dans le chapitre 1 nous introduisons quelques définitions utiles à la lecture du document.

Dans le chapitre 2 nous présentons un état de l'art sur les différents intergiciels permettant de gérer les ressources d'une grille. Après avoir classé les approches principales, nous étudions leurs caractéristiques en termes de système d'information, d'ordonnanceur, de support à l'exécution d'applications distribuées et de tolérance aux défaillances.

Dans le chapitre 3, nous présentons l'approche que nous avons suivie pour concevoir un système d'exploitation pour grille permettant de concilier l'exécution fiable et efficace d'une large gamme d'applications distribuées avec la simplicité d'utilisation et d'administration. Le système proposé est organisé autour de trois services principaux : le système d'information, le service d'allocation de ressources et le service de gestion d'application.

Dans le chapitre 4, nous décrivons le système d'information et le service d'allocation de ressources. Le système d'information a spécialement été conçu pour répondre aux exigences de la grande échelle, de la simplicité de découverte des ressources pour les utilisateurs et de la simplicité de maintenance pour les administrateurs. Le service d'allocation de ressources a été conçu en lien étroit avec le système d'information de façon à pouvoir allouer efficacement des ressources aux applications en prenant en compte les contraintes des applications comme la nécessité d'avoir un réseau d'interconnexion efficace entre les nœuds utilisés par les tâches.

Dans le chapitre 5, nous présentons le service de gestion d'application que nous avons conçus pour garantir une exécution efficace et fiable d'applications distribuées de calcul scientifique dans une grilles dont les nœuds sont volatiles.

Dans le chapitre 6, nous présentons une mise en œuvre d'un prototype fonctionnel de Vigne permettant de gérer les ressources d'une grille et d'exécuter des applications distribuées de façon fiable. Afin de valider notre approche, nous présentons également des expérimentations conduites à grande-échelle sur la plate-forme Grid'5000 et par simulation.

Dans le chapitre 7, nous présentons la conception et la mise en œuvre d'une extension à la plate-forme de simulation numérique SALOME afin de bénéficier du système Vigne pour la gestion des ressources d'une grille et de SALOME pour le pilotage des applications complexes.

Finalement, nous concluons par un bilan de nos contributions et sur les perspectives ouvertes par notre travail.

Chapitre 1

Définitions

Définition 1.1 (Machine physique) *Unité électriquement indépendante permettant d'exécuter des tâches. Une machine physique peut être une station de travail, un calculateur ou un super-calculateur.*

Définition 1.2 (Nœud d'un système distribué) *Nous appelons nœud, une ou un ensemble de machines physiques accessibles en un seul point par le réseau physique. Un nœud peut être une station de travail, un calculateur, un super-calculateur, une grappe de calculateurs exploitée par un gestionnaire de traitement par lots ou par un système à image unique.*

Définition 1.3 (Tâche) *Une tâche est une suite d'actions qui résultent de l'exécution d'un programme informatique.*

Définition 1.4 (Application) *Une application est un ensemble de tâches pouvant interagir pour parvenir à un but commun. Les tâches peuvent s'exécuter de façon parallèle ou séquentielle.*

Définition 1.5 (Application distribuée) *Une application distribuée est composée de tâches qui sont réparties sur plusieurs machines physiques.*

Définition 1.6 (Application composée de tâches communicantes) *Une application composée de tâches communicantes est une application distribuée dont les tâches s'échangent des données en cours d'exécution.*

Définition 1.7 (Ressource) *Nous appelons ressource une partie ou l'ensemble des caractéristiques de calcul d'un nœud. Par exemple, une tâche peut s'exécuter sur une machine biprocesseur en n'utilisant qu'un processeur et que 50% de la mémoire, nous dirons alors qu'elle utilise la moitié des ressources de ce nœud.*

Définition 1.8 (Découverte de ressources) *La découverte de ressources consiste à trouver des nœuds dans la grille qui possèdent les ressources nécessaires à l'exécution d'une application.*

Définition 1.9 (Allocation de ressources) *Nous appelons allocation de ressources l'opération de choix des nœuds découverts à l'issue d'une découverte de ressources et l'opération d'affectation d'une partie des ressources de ces nœuds en vue d'exécuter une application.*

Définition 1.10 (Co-allocation de ressources) *La co-allocation de ressources consiste à allouer des ressources simultanément aux tâches d'une partie d'une application. Cela est utile pour l'exécution d'applications composées de tâches communicantes.*

Définition 1.11 (Intergiciel) *Nous appelons intergiciel, la couche logicielle qui est positionnée entre le système d'exploitation des ressources et les applications des utilisateurs.*

Définition 1.12 (Application patrimoniale) *Une application est dite patrimoniale lorsqu'elle n'est disponible que sous forme binaire. Sa re-compilation ou la ré-édition de ses liens n'est pas possible.*

Définition 1.13 (SAN) *Un SAN (System Area Network) est un réseau d'interconnexion à très faible latence dédié à des nœuds se trouvant dans une même pièce comme les grappes de calculateurs. Des exemples de SAN sont Myrinet [21], Infiniband [16] ou Quadrics [29]. À titre indicatif la latence est de l'ordre de 2 à 3 μs et le débit peut aller jusqu'à 10 Gb/s.*

Définition 1.14 (LAN) *Un LAN (Local Area Network) est un réseau d'interconnexion dédié à des nœuds se situant dans un même bâtiment ou dans plusieurs bâtiments proches. La technologie LAN la plus couramment utilisée actuellement est Gigabit Ethernet. La latence d'un LAN est de l'ordre de 50 μs et le débit est de 1 Gb/s.*

Définition 1.15 (WAN) *Un WAN (Wide Area Network) est un réseau permettant d'interconnecter des nœuds séparés par de grandes distances géographiques (de quelques km à plusieurs milliers de km). Les latences varient avec la distance de quelques ms à quelques centaines de ms et les débits peuvent varier entre quelques Kb/s (dans le cas d'une connexion ADSL) et plusieurs dizaine de Gb/s (dans le cas d'une connexion dédiée aux hautes performances).*

Définition 1.16 (Organisation virtuelle) *Nous appelons organisation virtuelle le résultat du partage temporaire de moyens de calcul appartenant à plusieurs individus ou institutions pour la coopération à des projets multiples. Un ensemble de règles établi par tous les participants défini les conditions du partage des moyens de calcul.*

Chapitre 2

La gestion des ressources dans les grilles

2.1 Introduction aux grilles

Les grilles de calculateurs sont apparues afin de pallier le manque de puissance et le coût démesuré des moyens de calcul de type machines vectorielles ou parallèles. L'ancêtre des grilles est le *metacomputing* qui était vu comme le futur de l'informatique parallèle. Le *metacomputing* avait pour but d'utiliser un grand nombre de ressources distribuées et hétérogènes comme une machine parallèle.

Foster et Kesselman [85] ont introduit pour la première fois en 1998 la notion de grille dans le cadre du projet Globus. Le terme de grille est lié à l'analogie avec la grille du réseau électrique. Tout comme la disponibilité du courant électrique, la ressource de calcul dans la grille est disponible à partir d'une simple prise réseau. Une grille de calcul est vue comme le partage de ressources de calcul dans un environnement flexible et sécurisé par une collection dynamique d'individus et d'institutions.

Plusieurs modèles de grilles peuvent être défini en fonction du type des calculs devant être exécutés, du nombre de participants, du nombre et du type de ressources utilisées ou encore du degré de sécurité requis.

2.1.1 Les architectures de grille

Nous présentons dans cette partie les architectures communément utilisées pour l'interconnexion d'unités de calcul.

2.1.1.1 Les réseaux de stations de travail

Les réseaux de station de travail sont nés du constat suivant. Dans les institutions, grandes entreprises ou laboratoires de recherche, les stations de travail des employés sont sous-exploitées [116, 125]. Comme le montre la figure 2.1, l'idée est de les regrouper afin de les exploiter lorsqu'elles sont inutilisées en journée ou alors en dehors des horaires de bureau, pour effectuer des calculs. Les stations de travail sont interconnectées par un LAN. L'échelle

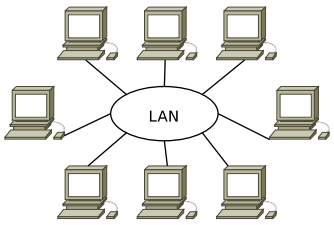


FIG. 2.1 – Réseau de stations de travail

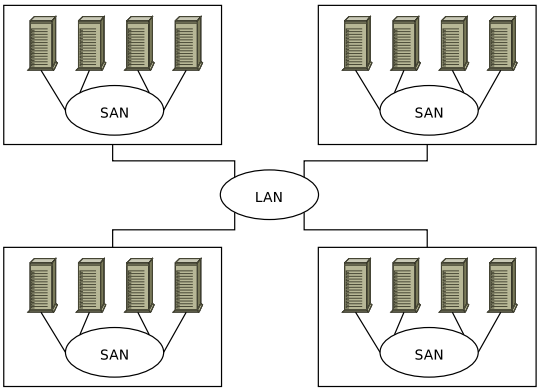


FIG. 2.2 – Fédération de grappes

des réseaux de stations de travail peut aller de plusieurs dizaines de nœuds à plusieurs milliers. Compte-tenu de leur mode d'exploitation, les nœuds d'un réseau de station de travail sont volatiles, d'autant plus qu'actuellement un grand nombre de stations de travail sont des machines portables.

2.1.1.2 Les fédérations de grappes

Les fédérations de grappes sont un regroupement de grappes au sein d'une institution. Comme le montre la figure 2.2, le réseau à l'intérieur d'une grappe est de type SAN et le réseau reliant les différentes grappes est de type LAN. L'échelle d'une fédération de grappes peut aller de quelques centaines à quelques milliers de nœuds. Les nœuds d'une fédération de grappes sont plus stables que des stations de travail. L'instabilité est dans ce cas uniquement dûe aux pannes ou aux opérations de maintenance.

2.1.1.3 Le calcul pair-à-pair

Le calcul pair-à-pair est né avec les premiers projets de recherche d'extra-terrestres dans l'univers où une quantité astronomique de données obtenues grâce à un radio-télescope devait être analysée [42]. Dans le calcul pair-à-pair, les ressources sont majoritairement des ordinateurs personnels connectés à Internet. L'échelle considérée est très grande, de l'ordre du million de nœuds. Les principales caractéristiques sont une très grande volatilité des nœuds et un nombre non négligeable de participants frauduleux.

2.1.1.4 Les grilles hétérogènes

Un dernier modèle de grille, qui est la combinaison des trois autres, concerne les architectures hétérogènes. Comme le montre la figure 2.4, une telle grille peut être constituée de super-calculateurs, de fédérations de grappes, de réseaux de stations de travail ou encore d'ordinateurs personnels. Ces ressources sont connectées par des réseaux SAN ou LAN au sein d'une même institution et via un WAN entre deux institutions. L'échelle considérée peut être très grande, certaines ressources peuvent être très stables et d'autres très volatiles.

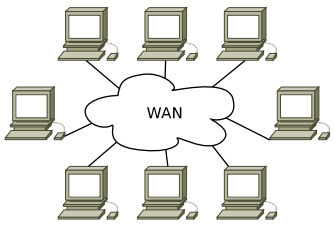


FIG. 2.3 – Le calcul pair-à-pair

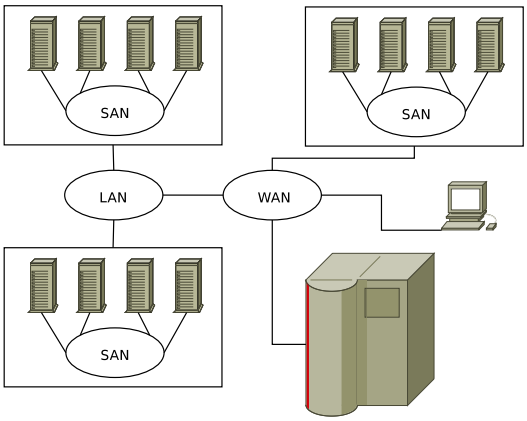


FIG. 2.4 – Grille hétérogène multi-site

Ce modèle de grille permet de décrire toutes les variantes de grille. Ainsi, c'est celui que nous retenons comme définition pour le terme grille.

2.1.2 Des exemples de grilles existantes

Nous présentons ici quelques exemples de grilles qui ont été déployées à grande échelle[1].

TeraGrid [33] est un projet financé par la NSF (*National Science Foundation*) dont l'objectif est de fournir la plus puissante grille de calcul scientifique au monde. Le projet est réparti sur 9 sites du territoire des États-Unis qui sont reliés par un réseau optique de 30 à 40 Gb/s. TeraGrid possède une capacité de calcul 102 Tflops et une capacité de stockage de 15 Pb. Les ressources de TeraGrid sont très hétérogènes en termes de calculateurs (PC multi-processeurs d'architecture Intel, SGI, PowerPC, ...), en termes de système d'exploitation (Linux, Unix propriétaires) et en termes de SAN. Les nœuds de TeraGrid forment une grille hétérogène multi-site.

EGEE (*Enabling Grids for E-sciencE*) [53] est un projet financé par la Commission Européenne dont l'objectif est la création d'une grille de production pour les scientifiques européens. EGEE compte plus de 20000 processeurs et une capacité de stockage de l'ordre de 5 Pb. EGEE utilise le réseau GÉANT2 (*Gigabit European Academic Network*) à 10 Gb/s. Les nœuds d'EGEE forment une grille hétérogène multi-site.

PlanetLab [27] est un projet atypique de grille dont l'objectif est de créer une grille planétaire. Actuellement, PlanetLab compte 753 nœuds répartis sur 363 sites. PlanetLab est une cible de choix pour l'expérimentation de technologies de type *pair-à-pair* ou de type *stockage distribué*. Les nœuds de PlanetLab forment une grille de type calcul pair-à-pair.

Grid'5000 [9, 57] est un projet national né d'une *Action Concertée Incitative* du ministère de la Recherche dont l'objectif est de créer une plate-forme expérimentale pour la recherche sur les grilles de calcul. La plate-forme Grid'5000 est actuellement composée de 9 sites répartis sur le territoire français. L'architecture de Grid'5000 est de type fédération de grappes. Actuellement, la plate-forme compte un peu plus de 2500 processeurs qui sont majoritairement d'architecture x86-64 (AMD Opteron) ou EMT64 (Intel Xeon). Grid'5000 s'appuie sur

[1] Les données présentées ici datent de février 2007.

le réseau RENATER [30] qui permet de relier les différents sites à 1 ou 10 Gb/s. De plus, il est possible de reconfigurer entièrement le système présent sur les machines en déployant un nouveau système d'exploitation.

ApGrid (Asia Pacific Grid) [2]. C'est une grille d'expérimentation qui comporte des ressources (environ 1600 processeurs) réparties sur les continents asiatique, océanique et américain. Même si les nœuds interconnectés sont des grappes, l'échelle de cette grille et la qualité du réseau d'interconnexion font que sa structure est plutôt un cas particulier de grille hétérogène multi-site qu'une fédération de grappes.

2.1.3 Les modèles d'applications pour la grille

Les applications dites réparties permettent de tirer profit des ressources d'une grille. Nous présentons dans cette partie différents modèles d'applications réparties.

2.1.3.1 Sac de tâches indépendantes

Certaines applications scientifiques comme les applications paramétriques, sont composées d'un grand nombre de tâches indépendantes. Typiquement, pour les applications paramétriques, le même programme est exécuté plusieurs fois avec un jeu de données différent.

2.1.3.2 Maître/travailleurs

Les applications de type maître/travailleurs permettent d'ordonnancer un grand nombre de tâches sur un nombre limité de ressources. Ce modèle est aussi adapté aux applications paramétriques où dans ce cas, le maître distribue les jeux de données aux travailleurs qui exécutent tous le même code. Ce modèle permet plus de finesse que le précédent. Il permet notamment de distribuer des tâches tant que le calcul n'a pas atteint un point de convergence. Le nombre de tâches à exécuter n'est dans ce cas pas connu à l'avance.

2.1.3.3 Applications parallèles

Les applications parallèles sont composées d'un ensemble de tâches plus ou moins communicantes selon leur degré de parallélisme. Elles sont donc particulièrement sensibles à la qualité du réseau sous-jacent. Ces applications sont souvent fondées sur le paradigme de communication par échange de messages tels que MPI [20] ou PVM [160]. L'opération, appelée parallélisation, consistant à adapter l'architecture d'une application pour qu'elle tire profit du parallélisme peut être complexe et peut nécessiter l'intervention d'un expert. D'autres environnements comme OpenMP [24] permettent de simplifier la phase de parallélisation en parallélisant automatiquement les boucles qui ont été anotées à l'aide de directives spécifiques dans le code des applications.

Les applications parallèles sont souvent utilisées pour résoudre des problèmes d'algèbre linéaire.

2.1.3.4 Applications distribuées

Les applications distribuées ont des contraintes moins fortes en ce qui concerne le réseau que les applications parallèles. Elles sont souvent fondées sur le paradigme d'invocation de méthodes distantes et utilisent des environnements comme CORBA [128] ou Java RMI [17]. Le paradigme composant, d'un niveau d'abstraction encore plus élevé, simplifie la programmation. Les environnements tels que CCA [46] ou encore CCM [127] permettent de créer des applications suivant le paradigme composant.

2.1.3.5 Couplage de codes

Certains problèmes scientifiques font intervenir plusieurs disciplines. C'est par exemple le cas lorsque l'on veut comprendre l'origine de perturbations climatiques telles que le phénomène *El Niño*. Il faut en effet modéliser plusieurs phénomènes comme l'océan et l'atmosphère [154]. Dans ce cas, des applications existantes et spécialisées pour chaque discipline sont couplées pour effectuer une simulation globale prenant en compte plusieurs physiques et garantissant une meilleure modélisation de phénomène étudié. Typiquement les codes couplés sont des applications parallèles qui requièrent un réseau sous-jacent rapide. Selon le niveau de couplage, les codes couplés peuvent communiquer intensément, et peuvent parfois nécessiter l'utilisation d'un réseau rapide pour ne pas dégrader les performances globales.

2.1.3.6 Composition de modèles

Finalement, tous ces modèles peuvent être combinés dans une méta-application appelée *workflow* [93]. Un *workflow* est l'enchaînement de plusieurs tâches (ou groupes de tâches) qui peuvent s'échanger des données. Ainsi, une tâche (ou un groupe de tâches) d'un *workflow* peut devoir attendre le résultat d'autres tâches (ou groupes de tâches) pour démarrer.

2.1.4 Couches logicielles sur les nœuds de la grille

Les nœuds d'une grille peuvent posséder différentes couches logicielles, et cela indépendamment de l'architecture de la grille. Cela est déterminé par leur mode d'exploitation hors de la grille.

Le mode d'exploitation le plus simple est celui fourni par un système d'exploitation de type Linux, Unix ou Windows. Dans ce cas les ressources sont utilisées en mode interactif. Les ressources utilisant ce mode d'exploitation sont les ordinateurs personnels, les stations de travail ou les grappes dites beowulf [3].

Lorsque les ressources sont des grappes, il est possible qu'elles possèdent un mode d'exploitation plus performant que le mode beowulf. Deux approches existent pour cela : les gestionnaires de traitement par lots et les systèmes à image unique.

Les gestionnaires de traitement par lots permettent de gérer la concurrence d'accès aux ressources en utilisant des files d'attente, éventuellement dotées de priorités. L'exécution de certaines tâches est ainsi différée en attendant la libération des ressources nécessaires. Il existe un grand nombre de gestionnaires de traitement par lots ayant chacun différentes fonctionnalités d'ordonnancement comme le *backfilling* [155], le *gang-scheduling* [78] ou encore le

co-scheduling [88]. Les systèmes propriétaires les plus utilisés en milieu industriel sont PBS Pro [1], Platform LSF [28] ou encore Sun Grid Engine [32]. Parmis les systèmes non propriétaires nous pouvons citer Torque [35] ou OAR [62].

Les systèmes à image unique [25, 49, 95, 124] donnent l'illusion que la grappe est un multi-processeur. Des systèmes comme MOSIX [49] mettent en œuvre un ordonnancement global des processus qui permet à la charge d'être répartie sur toute la grappe. D'autres systèmes comme Kerrighed [124] proposent en plus des fonctions avancées comme une mémoire virtuellement partagée qui permet une programmation simplifiée des applications, des flux de communication efficaces ou encore un mécanisme de points de reprise.

2.2 Les différentes approches logicielles pour exploiter une grille

De par leur nature, les grilles de calcul sont plus complexes à utiliser qu'une station de travail ou qu'une grappe. Cela est dû à la distribution des nœuds qui peuvent être situés dans des domaines d'administration différents. De plus, de l'échelle des grilles découlent deux caractéristiques principales des nœuds qui sont la volatilité et l'hétérogénéité. De nombreux projets se sont intéressés à la gestion des ressources d'une grille et ont proposé des systèmes, appelés intergiciels, pour en simplifier l'utilisation.

Dans cette partie, nous présentons différents types d'intergiciels, classés par rapport aux fonctionnalités qu'ils offrent.

2.2.1 Intergiciels de communication

Les intergiciels de communication visent à simplifier l'accès à des ressources réparties sur une grille en fournissant un moyen aux ressources de communiquer entre elles.

JXTA [18] de Sun Microsystems est un ensemble de protocoles pair-à-pair génériques qui permettent à n'importe quel nœud connecté au réseau de communiquer et de collaborer.

NaradaBrokering [130] a pour objectif de fournir un environnement de communication unifié pour différents paradigmes de programmation tels que les services Internet (*web services*), les services de grille (*grid services*) et les modèles pair-à-pair. NaradaBrokering offre deux méthodes de communication qui sont l'échange de messages et la publication/souscription.

PadicoTM [71] est un environnement de communication à hautes performances qui permet de multiplexer l'accès au réseau depuis plusieurs intergiciels. PadicoTM optimise les communications lorsque cela est possible en profitant d'un accès direct aux cartes de réseaux rapides et en ajustant les paramètres de la couche de transport. De plus, PadicoTM permet l'exécution d'applications réparties de part et d'autre de coupe-feu ou de réseaux isolés accessibles uniquement via un nœud frontal.

2.2.2 Intergiciels de gestion globale des ressources

Dans cette partie, nous présentons les approches logicielles pour gérer de façon globale les ressources d'une grille. Au minimum, ces intergiciels fournissent des mécanismes permettant de soumettre une application sur la grille, de trouver des ressources, de déployer une application et de récupérer les résultats produits.

2.2.2.1 Boîte à outils

Le projet Globus [84, 86] est né à la fin des années 1990 à l'Argonne National Laboratory et a pour objectif de construire une boîte à outils permettant d'utiliser de façon sécurisée une grille dont les ressources sont réparties à travers plusieurs domaines d'administration. Dans sa version 4, Globus est la mise en œuvre de référence de la norme OGSA (*Open Grid Services Architecture*) qui est proposée par l'Open Grid Forum (OGF) et dont l'objectif est de définir une interface qui permet l'inter-opérabilité entre les ressources d'une grille. Voyons les principaux constituants de la boîte à outils Globus. GSI (*Globus Security Infrastructure*) [169] est une infrastructure permettant d'authentifier les utilisateurs d'une organisation virtuelle qui est fondée sur l'utilisation de certificats X.509. GSI constitue la fondation de tous les autres services. MDS (*Monitoring and Discovery Service*) constitue le système d'information de Globus où toutes les ressources et leur état sont enregistrés. C'est en faisant une requête RSL (*Resource Specification Language*) que les utilisateurs peuvent interroger le MDS afin d'obtenir des ressources. GRAM (*Globus Resource Allocation Manager*) permet de soumettre des tâches sur des ressources distantes et d'en surveiller l'exécution. DUROC (*Dynamically Updated Request Online Co-allocator*) [70] est un service au-dessus de GRAM qui permet la soumission simultanée de multiples tâches, c'est-à-dire, la co-allocation de ressources. GridFTP (*Grid File Transport Protocol*) et GASS (*Global Access to Secondary Storage*) permettent respectivement de gérer les transferts de fichiers entre les ressources et de gérer les répliques de données pour optimiser les temps d'accès. La nature de l'architecture de Globus en fait un système très polyvalent mais complexe à utiliser. Globus ne fournit que les briques de base et c'est souvent à l'utilisateur qu'est laissé le soin d'adaptation pour des utilisations spécifiques comme par exemple la mise en œuvre d'un ordonnanceur de haut niveau fondé sur un modèle économique [60]. Toutefois, de par sa position privilégiée à l'OGF, Globus est presque considéré comme un standard *de facto* et constitue une fondation pour de nombreux travaux. Nous pouvons citer par exemple gLite [54], NorduGrid [75], GridLab [11] ou encore GridBus [61] qui sont des extensions de Globus visant à fournir des fonctionnalités de haut niveau pour l'exploitation des ressources d'une grille.

2.2.2.2 Systèmes intégrés

D'autres systèmes, plus simples à utiliser et disposant de fonctionnalités de plus haut niveau que Globus, permettent de prendre complètement en charge la gestion des ressources d'une grille. Ces systèmes ont pour objectif de masquer la distribution des ressources à l'utilisateur en offrant une vue unique de la grille. Ainsi, les fonctionnalités d'allocation de ressources, de sécurité, d'interface utilisateur, de transfert de fichiers sont au moins partiellement intégrées au sein de ces systèmes.

Systèmes génériques Condor [116] est un des premiers intergiciels intégrés qui est né à la fin des années 1980. Condor vise les architectures de type réseau de stations de travail et a pour objectif de récupérer les cycles inutilisés des nœuds de calcul. Les stations de travail forment un *Condor Pool* et un gestionnaire central est chargé de distribuer les tâches sur les stations. Les utilisateurs peuvent spécifier la nature des ressources requises pour leurs tâches grâce au

système ClassAdd (*Classified Advertisements*) [138]. Condor utilise un mécanisme de bac à sable pour l'exécution des applications afin d'éviter qu'un code malveillant ne compromette les nœuds utilisés. De plus, lorsqu'un nœud mobilisé par des tâches Condor est de nouveau utilisé par son propriétaire, les tâches qu'il exécutait sont déplacées sur un autre nœud via un mécanisme de point de reprise. Diverses évolutions de Condor ont été développées. Nous pouvons citer Flocking Condor [76] qui permet de fédérer plusieurs *Condor Pools*. Dans [59], les auteurs proposent une évolution du Flocking Condor en organisant automatiquement les *Condor Pools* dans un réseau pair-à-pair structuré. Linger-Longer [152] étend Condor pour supporter le vol de cycle avec un grain fin. Le problème avec le vol de cycle de Condor est que, dès qu'une utilisation même faible de la machine est détectée, les tâches sont suspendues. L'idée ici est de ne pas systématiquement suspendre ces tâches, mais plutôt de baisser leur priorité afin de ne pas gêner l'utilisateur.

Dans la lignée de Condor, Entropia [44] est un système permettant de fédérer les nœuds de type *poste de l'ingénieur* d'une entreprise. Entropia permet d'exécuter des tâches sur les nœuds de manière sécurisée pour les consommateurs et pour les fournisseurs de ressources. Pour cela Entropia utilise un bac à sable pour protéger le fournisseur de ressources et un mécanisme de cryptage de fichier pour protéger le consommateur de ressources.

UNICORE [147] est un projet européen dont l'objectif est la fédération de différents centres de calcul en Europe. UNICORE possède des fonctionnalités similaires à celles de Globus (sécurité, gestion des ressources, système d'information et gestion des fichiers) mais propose une interface d'utilisation conviviale qui intègre des fonctionnalités de haut niveau.

Legion [114, 66] a pour objectif d'offrir une vision unique des ressources de la grille. Pour cela, Legion propose un modèle objet où tout dans la grille (ressources de calcul, applications, fichiers) est représenté par un objet. Legion spécifie un certain nombre d'objets de bas niveau qui peuvent être modifiés ou étendus par les développeurs. Il est par exemple possible d'ajouter un support pour un nouveau gestionnaire de traitement par lots en dérivant la classe `HostClass` et en mettant en œuvre les nouvelles fonctionnalités souhaitées.

Vishwa [168] est un intergiciel intégré pour grille fondé sur des réseaux pair-à-pair qui permettent de tolérer la dynamicité des nœuds d'une grille. Vishwa fournit un composant pour la communication entre plusieurs tâches qui est fondé sur Distributed Pipes [107].

PUNCH (*Purdue University Network Computing Hub*) [108] est un projet de Purdue University dont l'objectif est d'utiliser la grille uniquement avec un navigateur internet. Via ce navigateur Internet, PUNCH fournit un moyen de déployer et d'utiliser des services sur la grille. PUNCH dispose d'un annuaire nommé ActYP (*Active Yellow Pages*) [151] qui permet de recenser les services déployés et qui possède de bonnes capacités de passage à l'échelle grâce à l'utlisation d'une méthode de réplication active des données.

Adage (*Automatic Deployment of Applications in a Grid Environment*) [111] est un projet développé au sein de l'équipe-projet PARIS de l'INRIA dont l'objectif est de fournir un mécanisme de déploiement pour les grilles de calcul. Adage propose un formalisme abstrait pour la description d'une application qui prend en compte la topologie du réseau et les applications fondées sur le modèle composant.

Zorilla [73] est un projet de Vrije Universiteit fondé sur les réseaux logiques pair-à-pair. Zorilla est un intergiciel complètement distribué qui possède toutes les fonctionnalités requises, comme l'ordonnancement ou le transfert de fichiers, pour l'exécution d'applications sur une

	Réseau de stations	Fédération de grappes	Calcul P2P	Hétérogène
Adage [111]	✔	✔	✔	✔
Condor [116]	✔	✘	✘	✘
Condor + Flocking [76, 59]	✔	✔	✘	✘
Entropia [44]	✔	✘	✘	✘
Globus [84]	✔	✔	✔	✔
Legion [114]	✔	✔	✘	✔
PUNCH [108]	✔	✘	✘	✘
UNICORE [147]	✘	✔	✘	✘
Vishwa [168]	✔	✘	✔	✘
Zorilla [73]	✔	✘	✔	✘

TAB. 2.1 – Architecture cible des intergiciels de gestion globale des ressources

grille. Grâce à son mécanisme d'ordonnancement appelé *flooding scheduling*, Zorilla est capable de tenir compte de la localité des ressources dans l'allocation.

Systèmes spécifiques Un certain nombre de projets sont spécifiques à un type d'application donné. Nous pouvons citer OptimalGrids [113] qui est un intergiciel dédié au déploiement d'applications de type *éléments finis*. OptimalGrids permet de paralléliser automatiquement un problème en fonction des ressources disponibles, en fonction d'indications présentes dans une base de connaissance et en tenant compte du ratio $\frac{calcul}{communications}$. P3 (*Personal Power Plant*) [156] est un intergiciel fondé sur JXTA dont l'objectif est de fournir une abstraction pour simplifier la programmation des applications parallèles fondées sur le passage de messages et les applications de type maître/travailleurs. Organic Grid [55] est un intergiciel permettant d'exécuter des applications composées de tâches indépendantes. Organic Grid reprend la métaphore de la colonie de fourmis où chaque nœud, représenté par un agent, est capable de s'adapter à son environnement et ne possède qu'une faible connaissance de l'environnement global.

2.2.2.3 Architectures cibles des intergiciels de gestion globale des ressources

Le tableau 2.1 présente les architectures de grilles pour lesquels sont conçus les intergiciels de gestion globale des ressources qui ont été étudiés dans cette partie.

Nous pouvons voir que dans l'état de l'art, les intergiciels de gestion globale des ressources ne sont pas adaptés à toutes les architectures de grilles. Néanmoins, les projets Adage et Globus peuvent être utilisés dans tous les contextes de grille mais ce sont les projets qui offrent le moins de fonctionnalités de haut niveau. Nous voyons dans la partie 2.2.4 que de nombreuses extensions à Globus qui concernent en particulier l'ordonnancement permettent d'augmenter le niveau de fonctionnalités en conservant sa polyvalence initiale.

2.2.3 Systèmes d'exploitation pour grille

Afin de donner une vision aussi parfaite que possible d'un *système à image unique*, certains projets s'autorisent à modifier directement un système d'exploitation afin d'en étendre les fonctionnalités. Des propriétés uniques peuvent ainsi être obtenues comme une transparence complète pour l'exécution des applications ou encore un support très avancé pour les opérations de points de reprise. Cette approche très puissante est toutefois plus complexe à déployer car les fournisseurs de ressources d'une grille ne sont pas forcément prêts à changer les systèmes d'exploitation afin de ne pas perdre les propriétés requises pour les exécutions locales comme le contrôle d'admission des tâches ou les politiques de priorités pour l'utilisation des ressources.

MOSIX [49] est un système à image unique pour grappe de calculateurs. Il permet d'effectuer de l'équilibrage de charge entre les nœuds en effectuant de la migration de processus. Une extension pour la grille [48] permet de profiter de la puissance de calcul d'une fédération de grappes mais le passage à l'échelle reste limité par le mécanisme de *deputy* utilisé dans MOSIX pour rediriger les entrées/sorties des processus déplacés.

9Grid [122] est fondé sur le système à image unique Plan9 [134] qui possède les propriétés de passage à l'échelle requises pour l'utilisation sur une grille de type fédération de grappes. 9Grid traite des problèmes d'authentification, de sécurité, de découverte de ressources et de gestion des données. 9Grid propose une solution pour l'inter-opérabilité avec d'autres intergiciels comme Globus en utilisant le protocole 9P. Il faut noter toutefois que Plan9 ne supporte pas nativement les applications du monde Linux ou UNIX.

XtreemOS [36] est un projet européen dont l'objectif est d'étendre le système Linux pour supporter les organisations virtuelles des grilles. XtreemOS vise les grilles dynamiques de très grande taille quelle que soit leur architecture.

2.2.4 Ordonnanceurs

De nombreux travaux se sont concentrés sur l'aspect ordonnancement de tâches et s'appuient sur des outils existants pour les autres fonctionnalités comme le système d'information, le transfert de fichiers ou encore la réservation des ressources.

SmartNet [89] est un projet dont l'objectif est l'ordonnancement des applications sur des ressources hétérogènes. Pour cela, SmartNet tient compte de la charge des machines mais aussi de leurs performances selon le type de calcul utilisé. SmartNet peut reposer sur Condor ou IBM LoadLeveler [15].

MyGrid [68] est un système permettant l'exécution d'applications de type sac de tâches. L'architecture de MyGrid comprend un ordonnanceur et une interface générique pour divers types de nœuds tels que ceux utilisant le GRAM de Globus. Contrairement à beaucoup d'approches, MyGrid est centré sur l'utilisateur (*user centric* en anglais). C'est-à-dire que lorsqu'un utilisateur soumet une application, c'est la machine utilisée pour la soumission qui exécute l'ordonnanceur. Ourgrid [43] est une extension de MyGrid fondée sur un modèle pair-à-pair pour partager des cycles au sein d'une organisation virtuelle. OurGrid permet de protéger les ressources utilisées en utilisant un mécanisme de bac à sable.

AppLeS (*Application Level Scheduling*) [55] est un projet définissant le processus de créa-

tion d'un ordonnanceur adaptable en proposant six étapes qui sont : la découverte de ressources, la sélection de ressources, la génération d'ordonnancements possibles, la sélection d'un ordonnancement, l'exécution de l'application et le ré-ordonnancement. Afin de permettre aux développeurs de ne pas systématiquement réécrire un ordonnanceur, AppLeS propose trois types d'ordonnanceurs qui sont APST (*AppLeS Parameter Sweep Template*) qui permet d'ordonnancer les applications paramétriques, AMWAT (*AppLeS Master Worker Application Template*) qui permet d'ordonnancer les applications de type maître/travailleur et SA (*Supercomputer AppLes*) qui permet d'ordonnancer les applications sur un super-calculateur à espace partagé. AppLeS repose sur les services fournis par des gestionnaires de ressources tels que Globus, Legion ou NetSolve.

Nimrod/G [60] est un ordonnanceur destiné à l'exécution de simulations paramétriques sur grille. Nimrod/G propose un ordonnancement fondé sur un modèle économique où les fournisseurs de ressources définissent un prix et les clients payent pour une exécution. Nimrod/G est construit sur Globus, notamment pour les services MDS, GRAM et GASS.

Condor/G [90] est une évolution du projet Condor. C'est un ordonnanceur fonctionnant audessus de la boîte à outils Globus. L'objectif est de fédérer plusieurs domaines d'administration en utilisant les mécanismes d'authentification de Globus. Les domaines fédérés peuvent être des *Condor pools* ou d'autres gestionnaires de tâches comme PBS et sont accédés via le GRAM de Globus.

SPHINX [117] est un ordonnanceur pour les grilles composées de ressources dynamiques. SPHINX propose des mécanismes d'ordonnancement avancés en utilisant les informations récupérées sur les ressources grâce à un système de supervision et en les stockant pour de futurs ordonnancements. Par exemple, si un nœud possède un nombre élevé de tâches qui ont été supprimées, il ne sera pas choisi en priorité. SPHINX s'appuie sur le système d'information MDS de Globus mais une interface générique a été définie pour fonctionner avec d'autres systèmes tels que Ganglia [118], MonALISA [126] ou Hawkeye [14].

KOALA [123] est un ordonnanceur pour la co-allocation de tâches fondé sur Globus (RSL, MDS, RLS, GRAM, GridFTP). KOALA possède des politiques d'ordonnancement permettant de minimiser les transferts de fichiers. Il permet ainsi de placer les composants applicatifs au plus près des fichiers d'entrée ou des répliques.

2.2.5 Intergiciels pour la programmation d'applications de grilles

Certains intergiciels sont dédiés à la programmation d'applications pour les grilles. Ils fournissent une API permettant la programmation d'applications selon un paradigme particulier qui abstrait la localisation des ressources.

Paradigme du passage de messages pour la grille MPICH-G2 [109] est une implémentation de MPICH qui permet de construire des applications pour la grille. MPICH-G2 est fondé sur Globus pour la sécurité et la soumission des tâches. MPICH-V [58] propose aussi une implémentation de MPICH, cette fois en prenant en compte la volatilité des ressources. MPICH-V propose plusieurs protocoles de tolérance aux défaillances qui sont : l'enregistrement de messages pessimiste (MPICH-V1 et MPICH-V2), l'enregistrement de messages causal (MPICH-V/causal) et la coordination globale non-bloquante (MPICH-V/CL) et bloquante

(MPICH-P/CL) suivant l'algorithme de Chandy-Lamport.

Paradigme de la mémoire virtuellement partagée pour la grille Des intergiciels permettent de créer des applications pour grille fondées sur le concept de la mémoire virtuellement partagée. C'est le cas par exemple des projets JuxMem [45], Mome [105] et Vigne [145], tous trois développés dans l'équipe-projet PARIS de l'INRIA.

Paradigme composant pour la grille ProActive [47] est un environnement développé dans l'équipe-projet OASIS de l'INRIA pour le développement d'applications fondées sur le paradigme composant Fractal. ProActive offre des mécanismes de communication de groupe, de support à la mobilité pour déplacer des composants afin de réaliser de l'équilibrage de charge ou encore d'abstraction des ressources avec le concept de *Virtual Node*. ProActive utilise un système d'information externe qui peut être RMIregistry [17], le MDS de Globus [84], LDAP [87] ou UDDI [81].

GridCCM [136], développé au sein de l'équipe-projet PARIS de l'INRIA, définit un modèle qui étend CCM [127] avec la notion de composants parallèles. GridCCM est tout particulièrement adapté au couplage de codes où des codes parallèles communiquent entre eux.

DG-ADAJ [40] est un intergiciel développé dans l'équipe PALOMA au LIFL permettant d'exécuter des applications utilisant le paradigme composant CCA [46] sur une grille de stations de travail. DG-ADAJ donne une vision SIU de la grille et offre des fonctionnalités pour l'exécution de *workflow* et pour l'équilibrage de charge.

Paradigme du calcul global Le paradigme du calcul global, ou paradigme du calcul pair-à-pair, consiste à utiliser un nombre de machines pouvant aller jusqu'à plusieurs millions réparties sur Internet pour l'exécution d'une application composée d'un grand nombre de tâches indépendantes.

BOINC (*Berkeley Open Infrastructure for Network Computing*) [41] est un projet de l'UC Berkeley qui vise à fournir une infrastructure pour le développement d'applications distribuées qui utilisent une grille de type *calcul pair-à-pair*. BOINC est particulièrement adapté aux applications de type maître/travailleurs, fournit des facilités pour la sécurité, la tolérance aux défaillances et l'ordonnancement. Un exemple d'application portée sur BOINC est SETI@home [42] dont l'objectif est d'analyser des données obtenues avec des radio-télescopes en vue de détecter une forme de vie extra-terrestre. En juin 2007, l'application SETI@home était exécutée sur un peu plus d'un million et demi de machines.

XtremWeb [63, 72] est un projet du LRI qui permet, comme BOINC, de développer des applications de calcul global. XtremWeb est adapté au paradigme maître/travailleurs et fournit une abstraction du concept d'appel de procédure à distance et propose des interfaces pour JavaRMI et XML-RPC. Dans [117], les auteurs proposent d'utiliser XtremWeb pour fédérer des *Condor Pools*. Dans ce cas, un *Condor Pool* est vu comme un travailleur pour XtremWeb.

HiPoP (*Higly distributed Platform Of comPuting*) [97] est une plate-forme de calcul global développée au LIFC. HiPoP permet d'exécuter des applications Java décrites par un graphe orienté acyclique.

Paradigme ASP Le paradigme ASP (*Application Service Provider*) consiste à utiliser un certain nombre de serveurs répartis sur des ressources et dédiés à la résolution de problèmes numériques.

Les intergiciels de type ASP masquent à l'utilisateur la localisation et le type des ressources et fournissent une API souvent fondée sur un modèle d'appel de procédures distantes pour le développement d'applications. Nous pouvons citer quelques exemples d'ASP : DIET (Distributed Interactive Engineering Toolbox) [64], Ninf/G [162], AROMA (scAlable ResOurces Manager and wAtcher) [132] ou encore GridSolve [172].

2.2.6 Bilan sur les différentes approches

Des nombreuses approches ont été proposées pour construire un intergiciel de grille. Certains systèmes visent à simplifier les communications entre les ressources d'une grille, d'autres prennent complètement en charge la gestion des ressources et d'autres fournissent des environnements de programmation pour la création d'applications de grilles.

Conceptuellement, ces approches sont complémentaires. En effet, il serait sensé d'utiliser un intergiciel de gestion de communication comme couche basse d'un intergiciel de gestion globale des ressources qui lui même serait utilisé en-dessous d'un intergiciel pour la programmation d'applications. Toutefois, pour des raisons d'implémentation, les interfaces des intergiciels présentés ne sont pas forcément compatibles. Cependant, certaines combinaisons sont possibles. C'est par exemple le cas de PadicoTM [71] qui peut être utilisé avec Adage [111] qui lui même peut être utilisé avec GridCCM [136], le tout étant conçu dans le projet Padico [26].

Nous pouvons noter que de nombreux travaux sont fondés sur Globus [84]. La raison principale est que Globus implémente les standards proposés l'OGF [23], ce qui garantit l'interopérabilité entre les différents travaux.

Finalement, il n'existe pas de travaux qui concilient généricité et fonctionnalités de haut niveau. Seule la combinaison de Globus et d'ordonnanceurs permet d'atteindre cet objectif, au prix toutefois d'une complexité de déploiement des différents intergiciels sur les nœuds de la grille.

2.3 Caractérisation des intergiciels pour grilles

Nous détaillons dans la partie suivante les caractéristiques des intergiciels pour grilles en étudiant leur système d'information, l'architecture de leur ordonnanceur, les types d'applications qu'ils supportent, leur support pour le dynamisme de la grille, la sécurité qu'ils offrent et leur interface d'utilisation.

2.3.1 Système d'information

Les intergiciels de grille utilisent un système d'information pour recenser les ressources qui composent la grille et éventuellement pour connaître l'état de ces ressources. L'état peut être simplement une information liée à la présence ou non de la ressource dans la grille, mais cela peut aussi être l'ensemble des caractéristiques de la ressource ou encore une information de charge.

2.3.1.1 Liste statique de ressources

Dans sa plus simple forme, un système d'information se réduit à une liste statique des ressources de la grille. C'est le cas par exemple des environnements MPI pour la grille comme MPICH-G2 [109] et MPICH-V [58] ou encore de l'outil de déploiement Adage [111].

Ce type de système d'information n'est pas adapté à la grande taille d'une grille puisque chaque ajout ou retrait de ressources nécessite la modification de la liste et le redéploiement de l'intergiciel.

2.3.1.2 Serveur central

Une autre forme de système d'information simple à mettre en œuvre consiste à utiliser un serveur central. Lorsqu'un utilisateur souhaite obtenir des ressources, il contacte le serveur central qui lui donne des informations à jour. UNICORE [147], Entropia [44], HiPoP [97] ou encore SmartNet [89] utilisent un serveur central comme système d'information. L'administrateur d'un site est en charge de l'ajout et la suppression des ressources dans la base de données du serveur. Dans les projets Condor [116] et XtremWeb [63], les fournisseurs de ressources ajoutent directement leurs ressources en effectuant une requête au serveur central et un *ping-pong* initié par le serveur permet de détecter le départ d'une ressource.

L'utilisation d'un serveur central pose un problème de fiabilité car sa défaillance induit la défaillance du système tout entier. De plus, l'utilisation d'un serveur centralisé peut devenir un goulet d'étranglement lorsque la taille de la grille est grande.

2.3.1.3 Serveurs dupliqués

Pour améliorer la fiabilité et le passage à l'échelle d'un serveur central, des approches de type réplication peuvent être utilisées. Le système PUNCH [108] utilise ActYP[151] pour son système d'information fondé sur des serveurs dupliqués qui permettent d'accélérer l'accès à l'information.

Si la taille de la grille est vraiment grande, il est nécessaire d'avoir un grand nombre de répliques pour éviter le risque de goulet d'étranglement. Cela est toutefois problématique car la bande-passante consommée pour assurer la cohérence entre les répliques augmente avec le nombre de répliques.

2.3.1.4 Serveurs hiérarchiques

Pour limiter le coût du maintien de la cohérence entre des serveurs dupliqués, une architecture hiérarchique peut être utilisée. Le projet Globus [84] propose une architecture hiérarchique pour son système d'information (MDS). MDS est fondé sur un annuaire réparti de type LDAP (*Lightweight Directory Access Protocol*). Beaucoup d'intergiciels pour grille sont fondés sur Globus. Par conséquent, ils héritent d'une architecture hiérarchique pour leur service d'information. C'est par exemple le cas de MyGrid [68], Nimrod/G [60], Condor/G [90] et KOALA [123]. DIET [64] est aussi fondé sur une architecture hiérarchique. Des agents principaux (*Master Agents*) sont chargés de répartir les requêtes des clients vers une hiérarchie

d'agents locaux (*Local Agents*) qui effectuent la soumission sur des serveurs possédant le service requis et qui tiennent à jour une liste des requêtes en attente pour leur sous-arbre.

Une architecture hiérarchique permet de résoudre le problème de passage à l'échelle mais la perte d'un nœud dans la hiérarchie induit la perte de tout le sous-arbre associé.

2.3.1.5 Découverte de ressources fondée sur des réseaux logiques

Afin de résoudre les problèmes liés à l'utilisation d'un serveur central, de serveurs répliqués ou d'une architecture hiérarchique, des approches complètement distribuées peuvent être envisagées. Ian Foster et Adriana Iamnitchi ont notamment étudié dans [83] la convergence entre la thématique du pair-à-pair et celle des grilles. En effet, les grilles et les systèmes pair-à-pair ont en commun le dynamisme, la grande échelle et l'hétérogénéité des ressources. Dans ce cas, l'interrogation du système d'information se réduit à une requête de découverte de ressources. Une étude similaire a été réalisée par Domenico Talia et Paolo Trunfio dans [161]. Initialement pensés pour les applications de partage de fichiers, les réseaux pair-à-pair constituent une technologie de choix pour ses bonnes propriétés de passage à l'échelle. Les architectures pair-à-pair peuvent être classées selon trois catégories qui sont liées à leur infrastructure.

Il existe des architectures pair-à-pair dites centralisées, comme le système Napster [22] par exemple. Dans ce cas, les clients désirant trouver une ressource (un fichier dans le cas de Napster) contactent un serveur central qui leur donne les informations concernant la localisation de la ressource (IP et port dans le cas de Napster). Ensuite les transferts se déroulent de pair à pair sans passer par le serveur.

Une autre forme d'architecture pair-à-pair permettant un meilleur passage à l'échelle est la forme hybride. Dans cette approche, le rôle de serveur central est divisé entre plusieurs serveurs dits super-pairs. Cette approche est utilisée par KaZaa [19] ou encore JXTA [18]. Dans [120] et [135], les auteurs ont appliqué le concept de super-pair au service d'information d'une grille suivant la norme OGSA.

Une dernière forme d'architecture pair-à-pair, cette fois totalement distribuée est dite décentralisée. Dans cette architecture tous les pairs ont un rôle identique. Il est encore possible de diviser ce type d'architecture en deux catégories selon la structure du réseau.

Les réseaux *non structurés* comme Gnutella [110] utilisent des mécanismes dérivés de l'inondation pour trouver des ressources dans le réseau. Chaque nœud connaît un nombre limité d'autres nœuds, appelés voisins, et lorsqu'il désire trouver une ressource, il envoie une requête à tous ses voisins. À la réception d'une requête, un nœud évalue s'il peut y répondre et fait suivre la requête à tous ses voisins. Et ainsi de suite jusqu'à une profondeur donnée. Pour cela un compteur appelé TTL (*Time To Live*) est incorporé dans les requêtes et il est décrémenté à chaque fois que la requête est envoyée, cela afin de limiter la profondeur d'inondation. Les systèmes Vishwa [168] et Zorilla [73] utilisent un réseau *non structuré* en guise de système d'information. L'inondation étant coûteuse du point de vue du réseau, des optimisations sont utilisées comme les marches aléatoires par exemple [94]. Contrairement à l'inondation classique, les marches aléatoires permettent de mieux contrôler le coût de la propagation des requêtes. Dans ce cas, un nœud n'envoie pas une requête à tous ses voisins, mais seulement à l'un d'eux choisi au hasard. Évidemment, une marche aléatoire ne suffit pas forcément à satisfaire la requête. Dans ce cas, il est possible d'effectuer plusieurs marches aléatoires successives

jusqu'à l'obtention du résultat désiré. Dans [99], les auteurs proposent d'améliorer le concept de marches aléatoires en utilisant des caches. Ainsi les requêtes sont par exemple envoyées aux nœuds qui ont le mieux répondu aux requêtes précédentes. Les réseaux *non structurés* sont particulièrement adaptés aux recherches de type multi-critères où les critères peuvent être définis sur des plages de valeurs.

Dans les réseaux *structurés*, par opposition aux réseaux *non structurés*, les nœuds sont insérés dans le réseau d'une façon bien déterminée. Par exemple dans Pastry [150] et Chord [157], les nœuds sont organisés autour d'un anneau logique ; dans CAN [140] ils sont organisés dans un cube ou encore dans espace de Voronoï avec VoroNet [52]. Les réseaux *structurés* permettent d'effectuer des recherches d'éléments dont le nom est connu bien plus efficacement qu'avec les réseaux *non structurés*, en utilisant un nombre limité de messages, souvent inférieur à $log(n)$ si n est le nombre de nœuds dans le réseau. Toutefois, les réseaux *structurés* ne sont pas adaptés aux recherches multi-critères. Des travaux sont cependant actifs pour combler cette lacune, nous pouvons citer Structella [65], Mercury [56], SWORD [129], GES [174] ou NodeWiz [51].

2.3.2 Type d'ordonnancement

Sans détailler les innombrables politiques d'ordonnancement qui existent dans l'état de l'art, nous étudions dans cette partie le niveau de centralisation du module d'ordonnancement des intergiciels de grille.

2.3.2.1 Ordonnancement centralisé

Le plus communément, les modules d'ordonnancement des intergiciels de grille sont mis en œuvre de façon centralisée. Avec une mise en œuvre centralisée, il y a un unique point d'admission des applications. Ensuite, la distribution des applications peut être effectuée à l'initiative du serveur central, c'est le mode *push*, ou alors à l'initiative des nœuds inactifs, c'est le mode *pull*.

Les projets Condor [116], Entropia [44], KOALA [123], SPHINX [165] ou encore HiPoP [97] sont fondés sur un ordonnanceur centralisé de type *push*. Le mode *push* permet de mettre en œuvre des politiques qui permettent d'ordonnancer des applications sur une grille d'une façon très efficace en décidant le moment précis où une application doit être exécutée. Ainsi, l'ordonnanceur peut ré-ordonner, par rapport à leur ordre de soumission, des applications pour minimiser le temps global d'exécution.

D'autres projets comme BOINC [41] ou XtremWeb [63] utilisent un ordonnancement de type *pull*. Avec le mode *pull* le serveur central n'a pas besoin de calculer d'ordonnancement car il est effectué au mieux dès qu'un nœud signale son inactivité. Cela permet un meilleur passage à l'échelle lorsque le nombre d'applications à exécuter et le nombre de nœuds sont très grands.

2.3.2.2 Ordonnancement hiérarchique

Une solution pour répondre au passage à l'échelle est d'utiliser une approche hiérarchique. Les projets AROMA [132], DIET [64] ou encore Legion [66] sont fondés sur cette approche.

Les tâches sont soumises à la racine de la hiérarchie et sont traitées par des agents d'ordonnancement situés dans les niveaux inférieurs de la hiérarchie.

L'approche d'AROMA consiste à suivre la hiérarchie imposée par une topologie de grille en fédération de grappes. DIET est plus souple puisqu'il offre la possibilité d'adapter la hiérarchie (pour ce qui concerne le nombre de niveaux et le nombre de feuilles d'un nœud) dans le but d'optimiser la vitesse d'ordonnancement d'un grand nombre de requêtes. Le projet Legion quant à lui fournit un modèle abstrait permettant de mettre en œuvre n'importe quel type de hiérarchie.

2.3.2.3 Ordonnancement coopératif

Les projets UNICORE [147], MOSIX [48] et Condor (doté de l'extension Flocking) [76] sont dotés d'ordonnanceurs coopératifs. Ils permettent de gérer des ressources sur plusieurs domaines d'administration. À l'intérieur d'un domaine d'administration, un ordonnanceur local est chargé de l'ordonnancement des applications sur les ressources des nœuds du domaine. Toutefois, en cas de surcharge des ressources locales, un ordonnanceur local peut se décharger de l'exécution de certaines applications en demandant aux ordonnanceurs des autres domaines s'ils peuvent accueillir ces applications.

2.3.2.4 Ordonnancement distribué

Une architecture distribuée est adaptée aux systèmes de très grande taille. Dans ce cas, les applications sont admises en n'importe quel point dans le système. Ainsi, autant d'ordonnanceurs que d'applications sont créés dans le système, chacun agissant de façon indépendante. Pour choisir le point de création d'un ordonnanceur, deux approches sont utilisées.

La première approche, utilisée par les projets OurGrid [43] et Zorilla [73], consiste à créer un ordonnanceur sur le nœud sur lequel l'application a été soumise.

Une seconde approche utilisée par le projet Vishwa [168] consiste à créer un ordonnanceur sur un nœud choisi aléatoirement dans la grille. Cette technique permet de résoudre le problème lié à la première approche si un grand nombre d'applications sont soumises depuis un même nœud.

2.3.3 Support pour l'exécution d'applications sur la grille

Dans cette partie, nous étudions le support fourni par les différents intergiciels de grille pour exécuter des applications.

2.3.3.1 Types d'applications supportées

Voyons les types d'applications présentés dans la partie 2.1.3 supportés par les différents intergiciels.

Applications non-communicantes Tous les intergiciels de grille supportent au moins les applications non communicantes de type application monotâche ou sac de tâches. Nous pouvons

citer Condor [116], SmartNet [89], MyGrid [68], Ourgrid [43] qui ne supportent que ce type d'applications.

Applications maître/travailleurs BOINC [41] et XtremWeb [63] sont dédiés à l'exécution des applications de type maître/travailleurs. Ces applications sont tout de même considérées comme communicantes, cela dit les communications ne sont pas effectuées entre travailleurs.

Applications communicantes Les implémentations grille de MPI comme MPICH-G2 [109] et MPICH-V [58] supportent naturellement les applications communicantes de type *parallèle*. Il en est de même pour GridCCM [136] et ProActive [47] qui sont des implémentations grille des paradigmes composants CCM et Fractal, et pour JuxMem [45], Mome [105] et Vigne [145] qui sont des implémentations grille du paradigme de mémoire virtuellement partagée.

De nombreux intergiciels supportent l'exécution d'applications communicantes puisqu'ils reposent sur des ressources qui utilisent un gestionnaire de traitement par lots. Ces intergiciels ne proposent donc pas de support spécifique pour les applications communicantes. C'est par exemple le cas de Globus [84] sans l'extension DUROC, UNICORE [147] et Adage [111]. Il en est de même pour certains intergiciels fondés sur Globus comme Nimrod/G [60] ou Condor/G [90].

Il faut noter qu'avec l'extension DUROC [70], Globus est capable d'exécuter des applications communicantes sans l'aide d'un gestionnaire de traitement par lots sous-jacent. DUROC fournit en effet des mécanismes de co-allocation de ressources. C'est-à-dire qu'il permet d'allouer plusieurs ressources simultanément à une application composée de tâches communicantes et de renseigner à chaque tâche la localisation des autres tâches de l'application pour que les communications puissent être établies. Une extension nommée Gangmatching [139] a aussi été apportée à Condor pour le support de la co-allocation. D'autres intergiciels supportent nativement la co-allocation de ressources, comme KOALA [123], Vishwa [168] ou Zorilla [73].

Finalement, les systèmes d'exploitation pour grille comme MOSIX [48], 9Grid [122] et XtreemOS [36] (qui est en cours de développement) supportent nativement l'exécution d'applications communicantes sur grille.

Applications composées La plate-forme HiPoP [97] supporte les applications composées d'un ensemble de tâches organisées sous la forme d'un graphe orienté acyclique. HiPoP ne supporte toutefois pas les applications communicantes et n'est donc pas adapté aux applications de couplage de codes. Plus généralement, les applications composées sont supportées par des moteurs externes appelés moteurs de *workflow*. Nous pouvons citer par exemple Triana [164], GWFE (*Gridbus Workflow Engine*) [10], DAGMan (*Directed Acyclic Graph Manager*) [7] ou GRMS (*GridLab Resource Management System*) [13].

2.3.3.2 Transparence pour l'exécution des applications

Le niveau de transparence pour l'exécution d'applications peut varier d'un intergiciel de grille à l'autre.

Certains intergiciels sont complètement transparents et ne requièrent pas la modification de l'application. C'est le cas par exemple de UNICORE [147], Entropia [44], Adage [111], Zorilla [73] ou encore MOSIX [49].

D'autres intergiciels requièrent une ré-édition des liens du code de l'application. C'est par exemple le cas de Condor [116] pour permettre aux applications d'être déplacées en utilisant les mécanismes de point de reprise de la librairie Condor. MPICH-G2 [109] requière aussi une ré-édition des liens de l'application pour bénéficier des fonctionnalités grille.

Des projets comme DUROC [70] ou Vishwa [168] requièrent l'ajout de primitives de synchronisation pour profiter des fonctionnalités de co-allocation.

Finalement, beaucoup de projets requièrent une écriture presque dédiée du code applicatif afin de bénéficier des fonctionnalités grille. C'est le cas par exemple de ProActive [47], GridCCM [136], XtremWeb [63], DIET [64] ou encore Legion [66].

2.3.4 Exécution fiable des applications

L'exécution fiable d'applications permet de garantir aux utilisateurs d'une grille que toute application soumise peut atteindre le terme de son exécution si son code ne contient pas d'erreur. Pour garantir d'exécuter de façon fiable une application, un intergiciel doit tout d'abord être capable de détecter une défaillance puis il doit être capable de réagir à cette défaillance.

2.3.4.1 Détection des défaillances d'applications

De nombreuses circonstances peuvent mener à la défaillance d'une application. Il peut tout d'abord y avoir une erreur dans le code de l'application qui entraîne sa défaillance (par exemple une erreur d'accès à la mémoire). Il peut aussi y avoir des défaillances liées au contexte d'exécution. Par exemple, une application peut utiliser le répertoire temporaire `/tmp` qui se retrouve plein au bout d'un moment. Dans ce cas l'application génère une erreur. La défaillance d'un nœud de la grille peut également conduire à la défaillance d'une application.

Généralement, les intergiciels de grille surveillent le premier processus lancé de l'application, typiquement un script *shell*. Ainsi, la terminaison de ce processus, suite à la défaillance d'un nœud ou suite à une erreur liée au contexte, est perçue comme une défaillance de l'application. C'est le cas notamment de Globus [84] avec le module GRAM et de tous les intergiciels fondés sur Globus, de Condor [116], d'UNICORE [147] ou encore d'XtremWeb [63].

D'autres intergiciels permettent une détection plus fine des défaillances. Les intergiciels Legion [66] ou GridLab/Mercury [12] sont notamment capables de détecter les défaillances de tous les processus d'une application, même ceux créés dynamiquement après le démarrage de l'application.

2.3.4.2 Réaction face à une défaillance d'application

Une solution simple pour réagir à la défaillance d'une application est de la redémarrer depuis le début sur d'autres nœuds. Les intergiciels XtremWeb [63], UNICORE [147], Organic Grid [55] proposent par exemple cette solution.

Condor [116] fournit un mécanisme de point de reprise qui peut être utilisé en cas de défaillance d'une application. Dans ce cas, l'application est redémarrée sur un autre nœud

depuis le dernier point de reprise capturé. Toutefois, cette approche n'est utilisable qu'avec les applications monotâche.

MPICH-V [58] propose plusieurs protocoles fondés sur l'enregistrement de messages ou la coordination globale pour exécuter des applications MPI en dépit des défaillances.

Vigne [145] propose également deux protocoles de cohérence de mémoire virtuellement partagée tolérants aux défaillances, assurant ainsi la continuité d'exécution de l'application.

Legion [66] propose, via son modèle objet, des mécanismes génériques pour stocker l'état des objets utilisés en vue d'une restauration ultérieure. La mise en œuvre de ces mécanismes est toutefois laissée à la charge du développeur.

BOINC [41] et MyGrid [68] peuvent exécuter la même tâche à plusieurs endroits afin d'assurer qu'au moins une exécution atteigne son terme. Dès qu'une exécution se termine, les autres exécutions sont annulées pour libérer les ressources.

2.3.5 Tolérance du système aux défaillances

La perte d'un nœud dans la grille peut avoir des incidences sur le fonctionnement du système de grille et sur le fonctionnement des applications en cours d'exécution sur la grille.

Plusieurs éléments d'un intergiciel peuvent être sensibles à la perte de nœuds. C'est par exemple le cas du système d'information, de l'ordonnanceur ou encore de l'interface de soumission des tâches.

Ainsi, des intergiciels comme Condor [116] ou XtremWeb [63] ne supportent pas la perte du coordinateur qui joue le rôle de service d'information et d'ordonnanceur. Il en est globalement de même pour toutes les approches centralisées.

Globus [84] peut survivre à des défaillances de nœuds qui toucheraient son système d'information (MDS) en utilisant le principe de réplication. LDAP est tout à fait adapté à cela. De plus, si une partie de MDS, non répliquée, est perdue lors d'une défaillance, seulement le sous-arbre associé est perdu, ce qui n'est pas forcément catastrophique si la défaillance n'a pas lieu tout près de la racine de l'arbre. PUNCH [108] utilise aussi le principe de réplication pour assurer la continuité du service malgré les défaillances.

MPICH-V [58] est conçu pour tolérer la volatilité des nœuds mais il repose sur un ensemble de ressources stables pour le stockage des messages notamment.

Des approches complètement distribuées tolèrent beaucoup mieux la perte de nœuds puisque les services ne sont pas centralisés sur un seul nœud. C'est par exemple le cas de Vishwa [168] ou Zorilla [73] qui sont fondés sur des approches pair-à-pair. Toutefois, le système NodeWiz [51] fondé sur une approche pair-à-pair ne supporte pas les défaillances de nœuds.

2.3.6 Protection des ressources mises à disposition dans la grille

Pour inciter les fournisseurs de ressources à partager leurs ressources dans la grille, l'intergiciel de grille peut garantir que l'intégrité des ressources est maintenue en dépit de l'exécution de programmes malicieux.

Cette garantie peut être obtenue en utilisant des mécanismes de type *bac à sable* où les applications qui s'y exécutent sont isolées du monde et en particulier, des ressources logicielles

de la machine hôte. Les intergiciels Condor [116], Entropia [44], XtremWeb [63] ou encore HiPoP [98] utilisent le principe du *bac à sable*.

D'autres travaux comme [80, 82, 137] sont fondés sur l'utilisation d'une machine virtuelle pour répondre à ce problème.

2.3.7 Interface d'utilisation

Il existe plusieurs types d'interfaces pour la connexion aux ressources d'une grille.

2.3.7.1 *Shell*

L'interface qui offre le plus de transparence est celle où l'utilisateur se connecte à la grille via un *shell*. Ce type d'interface est fourni par les systèmes d'exploitation pour grille tels que MOSIX [48], 9Grid [122] et XtreemOS [36].

2.3.7.2 Connexion à un serveur central

L'interface la plus simple à mettre en œuvre est celle qui consiste à se connecter à un serveur central pour soumettre une tâche. Cette approche est adoptée par exemple par les intergiciels KOALA [123], Nimrod/G [60], SmartNet [89] ou encore Condor [116]. Dans ce cas, un fichier descriptif de la tâche doit être créé par l'utilisateur et envoyé au serveur central. Une variante, adoptée par MyGrid [68] ou PUNCH [108], fournit aussi une interface de soumission sous forme d'un portail Web.

2.3.7.3 Connexion à différents serveurs

Certains intergiciels permettent aux utilisateurs d'effectuer des soumissions de tâches depuis plusieurs serveurs dans le système. C'est le cas par exemple pour UNICORE [147], NodeWiz [51] ou Condor doté de l'extension Flocking [76].

2.3.7.4 Connexion à tous les nœuds du système

Finalement, les intergiciels fondés sur des réseaux pair-à-pair comme Vishwa [168] ou Zorilla [73] permettent de soumettre des tâches au système depuis n'importe quel nœud.

2.3.8 Récapitulatif

Le tableau 2.2 présente un récapitulatif des caractéristiques des principaux intergiciels de grille.

Système d'information Cette caractéristique correspond à l'architecture du système d'information de l'intergiciel concerné.

Ordonnanceur Cette caractéristique détermine le niveau de centralisation de l'ordonnanceur de l'intergiciel concerné.

	Système d'information	Ordonnanceur	Support natif app. communicantes	Transparence	Survie du système après défaillance	Détection des défaillances des applications	Support pour la tolérance aux défaillances	Protection des ressources	Interface de soumission
9Grid [122]	hiérarchique	hiérarchique	oui	oui	oui	nc	non	non	shell
Adage [111]	liste	centralisé	oui	oui	non	non	non	non	nc
AppLeS [55]	hiérarchique	centralisé	non	oui	non	partielle	1	non	serveur central
BOINC [41]	centralisé	centralisé	non	non	non	oui	2	non	nc
Condor [116]	centralisé	centralisé	non	non	non	partielle	3	oui	serveur central
Condor/G [90]	hiérarchique	centralisé	non	oui	non	partielle	3	oui	serveur central
DIET [64]	hiérarchique	hiérarchique	non	non	oui	partielle	1	non	multiples serveurs
Entropia [44]	centralisé	centralisé	non	oui	non	partielle	1	oui	serveur central
Globus [84, 70]	hiérarchique	nc	oui	non	partielle	partielle	non	non	multitples serveurs
HiPoP [97]	centralisé	centralisé	non	oui	non	non	non	oui	serveur central
KOALA [123]	hiérarchique	centralisé	oui	oui	partielle	non	1	non	serveur central
Legion [66]	hiérarchique	hiérarchique	non	non	oui	oui	3	non	portail web
MOSIX [48]	distribué	coopératif	oui	oui	non	nc	non	non	shell
MPICH-G2 [109]	hiérarchique	nc	oui	non	oui	partielle	non	non	nc
MPICH-V [58]	liste	centralisé	oui	non	oui	oui	3	non	nc
Nimrod/G [60]	hiérarchique	centralisé	non	non	non	partielle	1	non	serveur central
OurGrid [43]	p2p	distribué	non	oui	oui	partielle	1	oui	tous les nœuds
ProActive [47]	centralisé	centralisé	oui	non	oui	oui	non	non	serveur central
PUNCH [108]	répliqué	hiérarchique	non	oui	oui	non	1	non	portail web
SmartNet [89]	centralisé	centralisé	non	oui	non	non	1	oui	serveur central
SPHINX [165]	centralisé	centralisé	non	oui	partielle	partielle	1	non	serveur central
UNICORE [147]	centralisé	coopératif	non	oui	non	non	1	non	multiples serveurs
Vishwa [168, 107]	p2p	distribué	oui	non	oui	non	1	non	tous les nœuds
XtremWeb [63]	centralisé	centralisé	oui	non	non	partielle	1	oui	serveur central
Zorilla [73]	p2p	distribué	non	oui	oui	partielle	non	non	tous les nœuds

TAB. 2.2 – Caractéristiques des différents intergiciels

Support natif d'applications communicantes Cette caractéristique spécifie si l'intergiciel supporte nativement les applications communicantes. Par nativement nous excluons les approches qui reposent sur un gestionnaire de traitement par lots et nous incluons les approches qui requièrent éventuellement une modification du code de l'application (comme Globus [84] avec DUROC [70] ou Vishwa [168] avec Distributed Pipes [107]).

Transparence Cela détermine si les applications doivent être modifiées ou non pour profiter de toutes les fonctionnalités offertes par l'intergiciel.

Survie du système après une défaillance quelconque Cela détermine si le système est capable de survivre après la défaillance de n'importe quel nœud, éventuellement celui hébergeant le système d'information, l'ordonnanceur ou encore l'interface de soumission. Après une telle défaillance, le système peut : fonctionner en mode dégradé (par exemple avec une partie des ressources inutilisables), fonctionner normalement ou ne plus fonctionner du tout.

Détection de défaillances d'applications Cela détermine si le système est capable de détecter une défaillance dans une tâche applicative. Dans le cas où cela est possible, la détection peut être complète ou seulement partielle si uniquement le processus père de la tâche est surveillé.

Support pour la tolérance aux défaillances Le support à la tolérance aux défaillances détermine si l'intergiciel possède des fonctionnalités permettant aux applications de s'exécuter correctement en dépit des défaillances. Ces fonctionnalités peuvent être des mécanismes fondés sur le redémarrage d'une application depuis le début (type 1), sur des exécutions dupliquées (type 2) ou sur des points de reprise (type 3).

Protection des ressources Cela détermine si l'intergiciel permet de protéger les ressources offertes dans le système, en utilisant par exemple un mécanisme de bac à sable.

Interface de soumission Cela détermine l'entité à laquelle les utilisateurs doivent s'adresser pour soumettre une tâche dans le système.

2.4 Conclusion

Nous avons présenté dans ce chapitre une étude sur la gestion des ressources d'une grille. Nous avons tout d'abord présenté différentes topologies de grilles, des exemples de grilles actuellement en fonctionnement, les modèles d'applications pouvant bénéficier de la puissance de calcul fournie par une grille et les couches logicielles présentes sur les nœuds d'une grille. Ensuite, nous avons présenté différentes approches pour exploiter et simplifier l'utilisation d'une grille de calcul. Finalement, nous avons caractérisé les intergiciels de grille en analysant : leur système d'information, leur support pour l'exécution d'applications sur la grille, l'impact de la défaillance de nœuds pour le système et les applications.

De cette étude, nous pouvons tirer différentes leçons. Tout d'abord, beaucoup d'intergiciels reposent sur une architecture centralisée ou hiérarchique qui suppose l'existence d'entités stables pour ne pas nuire au fonctionnement général du système. La supposition d'entités stables peut paraître utopique compte-tenu des spécifications des constructeurs sur le taux de défaillance du matériel informatique actuellement. Des approches fondées sur les technologies pair-à-pair répondent en partie à ce problème puisqu'aucune entité dans le système n'a la lourde responsabilité de tenir à jour un système d'information ou encore d'ordonnancer les applications sur la grille.

Seul le système Globus [84] est réellement polyvalent. Il fournit une boîte à outils permettant d'exécuter une large variété d'applications tout en adressant les problématiques de sécurité et en visant toutes les architectures de grilles. Cependant, son administration et son utilisation sont réellement complexes. De plus, Globus permet de mettre en œuvre des fonctionnalités de très haut niveau comme un ordonnanceur fondé sur un modèle économique par exemple, mais cela n'est pas fourni en standard. D'autres systèmes sont beaucoup plus simples à utiliser et mettent en œuvre des fonctionnalités de très haut niveau, mais sont dédiés à un type d'applications bien particulier. C'est par exemple le cas d'OptimalGrids [113] qui permet de paralléliser automatiquement des applications de type *éléments finis* et de les exécuter sur une grille.

Afin de profiter de fonctionnalités avancées comme la co-allocation de tâches ou encore un support pour la tolérance aux défaillances des applications, les applications requièrent souvent une ré-édition de liens, un ajout de primitives dans le code ou même une écriture spécifique. Cela est problématique car de nombreuses applications de calcul scientifique ont été écrites il y a de nombreuses années et ne peuvent plus être compilées sur les systèmes actuels. Ainsi, seuls les binaires sont exploitables.

Peu d'intergiciels proposent un support complet pour la détection des défaillances d'une application. De plus, lorsqu'une défaillance est détectée, les intergiciels ne prennent pas systématiquement de mesures pour que l'exécution se déroule convenablement. Ainsi, la fiabilité de l'exécution de l'application est souvent laissée à la charge des utilisateurs.

Pour conclure, les intergiciels actuels ne concilient pas complètement : simplicité d'utilisation et d'administration, exécution des applications distribuées sans modification des codes, robustesse dans un environnement de grille dynamique de grande taille et sans entité stable. Toutefois, OurGrid [43], Vishwa [168] et Zorilla [73], grâce à leur architecture fondée sur des réseaux pair-à-pair, semblent les plus aptes à fédérer les ressources d'une grille dynamique de grande taille. Parmi ces trois projets, Vishwa et Zorilla sont les seuls à fournir un support pour l'exécution d'applications communicantes.

Chapitre 3

Conception d'un système pour l'exploitation des ressources d'une grille de grande taille

Au regard de l'état de l'art sur la gestion des ressources de calcul dans les grilles, nous présentons maintenant les travaux que nous avons menés durant cette thèse. Nous commençons par décrire les objectifs que nous nous sommes fixés pour combler certaines lacunes de l'état de l'art dans le cadre de la gestion des ressources dans les grilles. Ensuite, nous détaillons l'approche que nous avons suivie pour atteindre ces objectifs. Finalement, nous présentons les éléments qui constituent l'architecture de notre système.

3.1 Objectifs

Nous présentons dans cette partie les objectifs de nos travaux relatifs à la conception d'un intergiciel pour l'exécution d'applications distribuées dans une grille de grande taille.

3.1.1 Grande échelle

L'intérêt majeur des grilles de calcul est lié à la grande puissance de calcul qu'elles peuvent fournir. Cela implique toutefois la participation d'un grand nombre de nœuds qui sont en général répartis sur plusieurs sites géographiques. La grande échelle induite par le nombre de nœuds participants et la distance géographique pouvant séparer les nœuds est sans aucun doute la caractéristique la plus contraignante dans l'utilisation des grilles.

Dans cette partie, nous détaillons les différentes sources de complexité liées à la grande échelle d'une grille que nous souhaitons traiter afin de proposer un système pouvant fédérer les ressources des divers types de grilles présentés dans la partie 2.1.1.

3.1.1.1 Nombre de ressources et distance géographique

Notre système de gestion de ressources doit pouvoir fédérer un grand nombre de ressources, de l'ordre de plusieurs milliers de nœuds de grille, un nœud de grille pouvant être une station de travail, un calculateur ou une grappe de calculateurs exploitée par un gestionnaire de traitement par lots ou un système à image unique.

De plus, le système doit tenir compte de la distance géographique qui sépare potentiellement les ressources, notamment pour les effets induits par des latences qui peuvent être supérieures à 100 ms.

3.1.1.2 Hétérogénéité des ressources

Plus le nombre de ressources fédérées est grand, plus la probabilité que les nœuds soient composés de ressources différentes est grande. Ainsi, deux types d'hétérogénéité existent : l'hétérogénéité matérielle et l'hétérogénéité logicielle.

Notre système de gestion de ressources doit être capable de fédérer efficacement des nœuds ayant des architectures processeur différentes, des quantités de mémoire différentes, des cartes réseaux différentes ou encore des supports de stockage de tailles et de performances différentes.

De plus, le système doit s'accommoder des nœuds exploités par différents systèmes d'exploitation et différents gestionnaires de traitement par lots.

Finalement, toutes les nœuds ne possèdent pas une installation identique et des versions différentes des librairies peuvent être installées. Par exemple, si une application requiert une version déterminée d'une librairie dynamique pour fonctionner, le système de gestion de ressources devra allouer uniquement des nœuds possédant la version adéquate de la librairie dynamique.

3.1.1.3 Comportement dynamique des nœuds

La grande échelle induit inéluctablement de nombreux ajouts ou retraits de nœuds. Ce comportement dynamique peut être dû à des défaillances des nœuds [121, 115] ou à des ruptures de liens réseau. Il peut aussi être dû à des déconnexions volontaires, dans le cas par exemple où un administrateur souhaite effectuer une mise à jour du système d'exploitation ou encore dans le cas où les nœuds sont partagés seulement une partie de la journée comme dans les réseaux de stations de travail. Plusieurs études comme [153, 96, 158, 69] ont été réalisées dans le cadre des réseaux pair-à-pair d'échange de fichiers sur les caractéristiques de la participation des utilisateurs. Il en ressort que la participation est de l'ordre de la dizaine de minutes en moyenne. Même si dans un contexte de grille, les ressources ont un comportement beaucoup moins dynamique, la tendance est tout de même valable. Si des stations de travail composent la grille, il est probable que la durée de participation des nœuds soit de l'ordre de quelques heures par jour seulement.

Le système de gestion de ressources doit donc être tolérant à des défaillances simultanées de nœuds. Nous ne souhaitons pas considérer d'entités stables qui pourraient héberger des éléments vitaux du système tels que le système d'information ou l'ordonnanceur comme c'est le cas avec UNICORE [147] ou XtremWeb [63] par exemple.

3.1.2 Simplicité

Un facteur sans doute essentiel pour la démocratisation des grilles dans le contexte industriel est la simplicité, tant pour les administrateurs que pour les utilisateurs.

3.1.2.1 Simplicité pour les administrateurs et les fournisseurs de ressources

Le système doit pouvoir être installé simplement sur les ressources sans nécessiter la lourde mise en place de nombreux services comme c'est le cas par exemple pour Globus [84].

Un pré-requis pour que les fournisseurs de ressources partagent avec confiance leurs ressources dans la grille est que le système ne soit pas trop intrusif. Il n'est donc pas vraiment souhaitable de modifier le système existant sur les nœuds. Dans le cas des grappes munies d'un gestionnaire de traitement par lots, il est impératif de conserver ce mode d'exploitation pour que des tâches puissent être lancées localement, sans passer par l'intergiciel de grille.

Finalement, il est souhaitable que l'enregistrement des nœuds qui participent à la grille soit automatique. En effet, dans un contexte où les nœuds de type *station de travail* participent à la grille, un enregistrement automatique soulage l'administrateur de nombreuses opérations. Ainsi, à l'opposé d'un système comme Globus [84] où l'enregistrement dans un annuaire des caractéristiques des nœuds mis à disposition est laissé à la charge des administrateurs, nous souhaitons que ces caractéristiques soient automatiquement extraites du système d'exploitation des nœuds.

3.1.2.2 Simplicité pour les utilisateurs

Nombre d'utilisateurs des ressources de calcul ne sont pas spécialistes en informatique, et encore moins en systèmes répartis. Il est donc essentiel de leur fournir des outils simples pour qu'ils profitent efficacement des ressources de la grille.

Tout d'abord, la distribution des ressources doit être occultée aux utilisateurs. Ainsi, pour exécuter une application sur la grille, un utilisateur doit seulement spécifier le type et le nombre de ressources dont il a besoin sans se soucier de la localisation ou du nom des machines.

Ensuite, l'utilisateur doit pouvoir décrire simplement le type de ressources requis pour l'exécution de ses applications. Il doit pouvoir également décrire simplement les applications complexes qui incluent des dépendances ou du parallélisme entre tâches.

3.1.3 Support d'une large variété d'applications

Nos travaux s'inscrivent dans le contexte du calcul à haute performance. Les ressources de calcul sont principalement dédiées à des applications de simulation numérique liées à un ou plusieurs domaines scientifiques. Par conséquent nous souhaitons apporter un support à l'exécution des applications distribuées qui sont utilisées en simulation numérique, celles présentées dans la partie 2.1.3, c'est-à-dire : les applications de type *sac de tâches*, les applications de type *maître/travailleurs*, les applications de type *parallèle*, les applications de type *couplage de codes* et les applications de type *workflow*.

3.1.4 Exécution fiable des applications

Pour encourager l'utilisation de la grille, les opérations liées à l'exécution d'applications doivent être limitées. En particulier, la volatilité des nœuds d'une grille peut impliquer des interruptions dans l'exécution des applications.

Le système doit donc garantir la fiabilité de l'exécution des applications des utilisateurs en fournissant les mécanismes nécessaires pour qu'une application lancée sur la grille arrive au terme de son exécution dans un temps raisonnable.

3.1.5 Exécution efficace des applications

Pour être exécutées dans un temps minimum, les applications doivent pouvoir utiliser pleinement les ressources qui leurs sont allouées. Par conséquent, si plusieurs applications sont exécutées simultanément sur un ensemble de nœuds, les applications doivent être réparties au mieux sur ces nœuds pour limiter les exécutions concurrentes sur un même nœud.

D'autre part, les applications de type *parallèle*, *couplage de codes* ou *workflow* peuvent être composées de tâches communicantes. Pour exécuter efficacement ces applications, le système de grille doit être capable de placer intelligemment les tâches qui communiquent entre-elles sur des nœuds de calcul qui possèdent une interconnexion réseau à faible latence pour limiter le coût des communications.

3.1.6 Sécurité

Les grilles de calcul introduisent de nombreux problèmes liés à la sécurité comme l'authentification des utilisateurs, la protection des ressources ou encore la confidentialité des données. De nombreux travaux comme CAS [133] ou VOMS [38] s'intéressent à ces problématiques. Nous les avons toutefois écartées de notre champ d'étude.

3.2 Approche

Dans cette partie nous exposons notre approche pour concevoir un intergiciel pour l'exécution d'applications distribuées dans une grille de grande taille qui permet d'atteindre les objectifs que nous avons énoncés dans la partie 3.1.

Dans un premier temps, nous présentons notre approche fondée sur la distribution des services système pour concevoir un intergiciel capable de fédérer les ressources d'une grille de très grande taille. Ensuite, nous présentons la généricité de notre approche pour la fédération des ressources des grilles de structures différentes et pour exécuter une large gamme d'applications. Finalement nous détaillons les concepts permettant d'exécuter de façon simple, efficace et fiable des applications distribuées sur une grille.

3.2.1 Système complètement distribué pour les grilles de grande taille

La grande taille d'une grille peut poser un certain nombre de problèmes à un système chargé de gérer les ressources. En effet, un gestionnaire de ressources de grille doit être capable de supporter un grand nombre d'utilisateurs qui peuvent soumettre de nombreuses applications sur de

nombreux nœuds de calcul. L'approche que l'on peut suivre naturellement consiste à exécuter, sur une machine, des services qui permettent la soumission d'applications sur un ensemble de nœuds préalablement enregistrés. Cette approche centralisée peut devenir un goulet d'étranglement dès lors que le nombre de nœuds, d'utilisateurs ou d'applications est grand. Pour cela, nous proposons de concevoir un intergiciel de grille de façon complètement distribuée. Notre approche pourra être comparée aux intergiciels fondés sur une architecture distribuée tels que OurGrid [43], Vishwa [168] ou encore Zorilla [73].

Pour construire un système complètement distribué et adapté aux grilles de grande taille nous proposons de fonder l'ensemble des services du système sur des réseaux pair-à-pair et sur le concept d'ordonnancement distribué.

3.2.1.1 Fondations pair-à-pair

Les réseaux pair-à-pair sont intrinsèquement adaptés aux systèmes distribués et dynamiques de grande taille. La communauté scientifique a adopté leur utilisation dans un contexte de grille comme le montrent par exemple [83] et [161].

Pour la construction d'un intergiciel fédérant les ressources d'une grille dynamique et de grande taille, les réseaux pair-à-pair décentralisés sont tout à fait adaptés compte-tenu de leurs propriétés de passage à l'échelle et de leur aptitude à survire dans un environnement composé de nœuds volatils. Grâce à leurs propriétés d'auto-organisation et d'auto-réparation, les réseaux pair-à-pair constituent les fondations idéales pour la construction de systèmes distribués ne nécessitant pas l'intervention d'un administrateur pour gérer l'ajout, la suppression ou la défaillance de nœuds dans le système.

Nous présentons maintenant deux types de réseaux pair-à-pair décentralisés qui permettent d'obtenir les propriétés nécessaires aux fondations d'un ensemble de services pour l'exécution d'applications sur une grille de grande taille composée de nœuds hétérogènes et volatils.

Réseau pair-à-pair structuré Louis Rilling a proposé dans sa thèse [143] de fonder un intergiciel de grille, en l'occurrence le système Vigne, sur un réseau pair-à-pair structuré de type Pastry [150]. Voyons ce qu'apportent les réseaux pair-à-pair structurés à un système comme Vigne.

Tout d'abord, chaque nœud dans le système possède un identifiant unique. Un nœud est capable de communiquer avec un autre nœud en connaissant seulement son nom, même s'il ne connaît pas sa localisation physique. Cela est rendu possible grâce aux capacités de routage dans les réseaux pair-à-pair structurés. Dans Pastry [150], le routage est effectué en moyenne en un nombre de $log(n)$ sauts si n est le nombre de nœuds dans le réseau. Les réseaux pair-à-pair structurés sont tout à fait adaptés à la mise en œuvre de tables de hachage distribuées (THD). Une THD permet de répartir un ensemble d'éléments sur un système distribué. Chaque élément dans le système est appelé *clé*. Dans une THD, chaque nœud est responsable d'un ensemble de clés qui peuvent représenter n'importe quel objet dans le système comme une tâche d'une application. Chaque clé possède un identifiant unique dans le même espace de désignation que les nœuds et chaque clé est gérée par le nœud qui possède l'identifiant numérique le plus proche. La figure 3.1 montre l'organisation des nœuds d'un réseau Pastry et des clés associées à ces nœuds. Nous pouvons noter que les nœuds, et par conséquent les clés, sont organisés autour

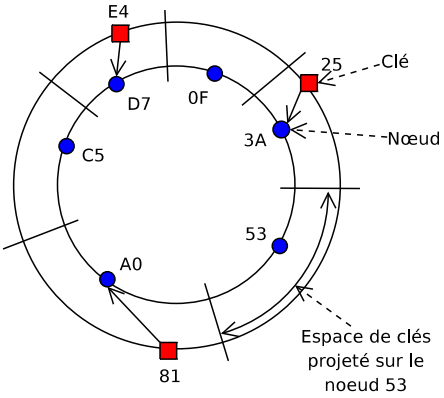


FIG. 3.1 – Organisation d'un réseau Pastry

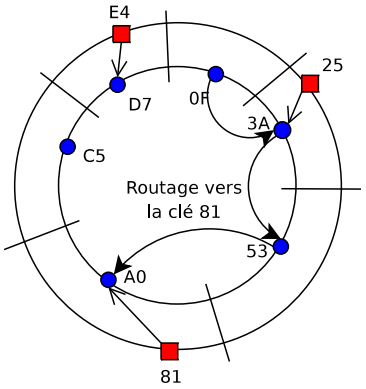


FIG. 3.2 – Routage dans un réseau Pastry

d'un anneau logique. La figure 3.2 montre un exemple de routage dans le même réseau où le nœud `0F` souhaite communiquer avec une entité possédant la clé `81`. Dans ce cas, la requête arrive par routage au nœud responsable de la clé `81`, c'est-à-dire le nœud `A0` car c'est celui qui a l'identifiant numérique le plus proche de `81`. Le routage dans un réseau pair-à-pair structuré rend la désignation des entités du système distribué indépendante de leur localisation. De plus, il permet la recherche d'entités à partir de leur nom logique avec un faible coût du point de vue du réseau.

Une propriété intéressante des THD est leur capacité à pouvoir gérer efficacement des répliques d'entités dans le réseau structuré. Comme le montre [144], l'association d'une THD et d'un mécanisme de diffusion de type *multicast atomique* permet la réplication active des entités du système sans qu'il n'y ait besoin de coordination de type *consensus* entre les répliques. Transposée au contexte de la construction d'un intergiciel de grille, la réplication active d'entités dans un réseau pair-à-pair structuré permet d'envisager l'élaboration de services autoréparants et donc fiables. Nous détaillons l'application de la duplication active pour construire un service fiable dans la partie 5.1.1.3.

L'abstraction de la localisation physique d'une entité dans un réseau pair-à-pair structuré permet d'abstraire la répartition des différentes composantes d'un service réparti sur une grille. De plus, la duplication active d'entités dans la THD permet à chaque composante d'un service réparti d'être tolérant à la volatilité des nœuds de façon transparente. La conjonction de ces deux propriétés permet de construire des services répartis et hautement disponibles qui sont adaptés à la grande taille et à la volatilité des nœuds d'une grille tout en permettant de masquer l'architecture répartie.

Dans le chapitre 5, nous présentons un service de gestion d'application conçu sur cette base.

Réseau pair-à-pair non structuré Si les réseaux pair-à-pair structurés sont tout à fait adaptés à la recherche d'entités dont le nom est connu, ils le sont beaucoup moins lorsque le nom n'est pas connu et que ces entités doivent être recherchées en fonction de certains critères. Comme nous l'avons vu dans la partie 2.3.1, certains projets comme [65, 56, 129] tentent d'utiliser un

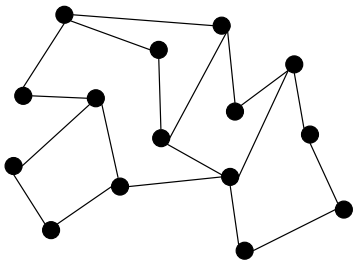


FIG. 3.3 – Exemple d'organisation de réseau non structuré

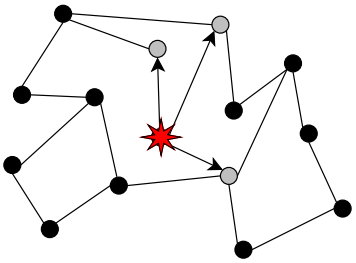


FIG. 3.4 – Inondation dans un réseau non structuré – profondeur 1

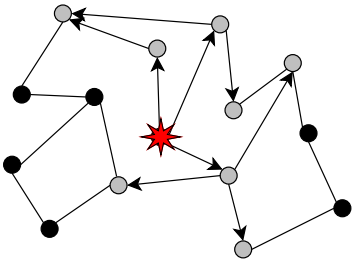


FIG. 3.5 – Inondation dans un réseau non structuré – profondeur 2

réseau structuré pour des recherches fondées sur un ensemble de critères mais la complexité de mise en œuvre est grande et la maintenance requiert une bande passante importante.

En revanche, les réseaux pair-à-pair non structurés sont naturellement adaptés aux recherches d'entités fondées sur un ensemble de critères, dont le nombre évolue dans le temps et dont la valeur peut être définie sur un intervalle. Comme le montre la figure 3.3, les nœuds d'un réseau non structuré sont organisés sans structure particulière, contrairement aux réseaux structurés où les nœuds sont organisés en fonction de leur nom par exemple. Ainsi, chaque nœud est connecté à un ensemble de voisins. Dans notre exemple, les connexions sont symétriques, mais ce n'est pas une obligation.

Lorsqu'un nœud souhaite rechercher une entité dans le réseau non structuré, plusieurs méthodes dérivées du concept d'inondation peuvent être utilisées. Dans le cas de l'inondation simple, un nœud envoie une requête de recherche à tous ses voisins et tous ses voisins envoient cette requête à leurs voisins, ainsi de suite jusqu'à une profondeur déterminée. La figure 3.4 montre la propagation d'une requête de recherche initiée par le nœud représenté par une étoile avec une profondeur de 1. De même avec la figure 3.5 pour une profondeur de 2. Sur ces figures, les nœuds ayant reçu la requête sont coloriés en gris. La requête de recherche qui est propagée de nœuds en nœuds peut contenir un ensemble de critères qui sont évalués sur chaque nœud. Si les caractéristiques d'un nœud correspondent à l'ensemble des critères, alors ce nœud répond à l'émetteur.

L'utilisation d'un réseau non-structuré permet de rechercher des nœuds d'un système distribué à partir d'une description fondée sur un ensemble de critères. Cela est tout à fait adapté à la fondation du système d'information d'une grille composée de nœuds hétérogènes. Rechercher une nœud d'un certain type convenant à l'exécution d'une application revient à faire une découverte de ressources sur un réseau pair-à-pair non-structuré où l'on propage une requête contenant une description multi-critères des ressources nécessaires.

Doté de mécanismes de composition tels que Scamp [92], un réseau pair-à-pair non structuré permet de construire un système d'information tolérant à la volatilité des nœuds grâce à des propriétés d'auto-organisation et d'auto-réparation. Cela est un atout supplémentaire qui permet à l'administrateur de s'affranchir d'opérations nécessaires à l'ajout ou au retrait de ressources dans une grille.

Dans le chapitre 4, nous étudions la conception d'un système d'information pour les grilles

de grande taille fondé sur l'utilisation d'un réseau pair-à-pair non structuré et sur l'optimisation d'un protocole de découverte de ressources favorisant la découverte de ressources libres et minimisant les coûts liés à l'utilisation du système.

3.2.1.2 Ordonnanceur distribué

L'ordonnanceur dans un intergiciel de grille est une partie importante d'un intergiciel de grille puisqu'il est chargé de répartir l'ensemble des applications soumises sur l'ensemble des ressources. Lorsque la taille de la grille et le nombre d'applications soumises sont grands, un ordonnanceur centralisé peut rapidement devenir un goulet d'étranglement et un point critique sensible aux défaillances.

Pour pallier ce problème, nous proposons un ordonnanceur complètement distribué. Dans notre approche, chaque fois qu'une application est soumise à l'intergiciel de grille, un agent d'ordonnancement que nous appelons *gestionnaire d'application* est créé sur un nœud de la grille aléatoirement choisi.

Grâce à l'utilisation sous-jacente des réseaux logiques structurés, chaque gestionnaire d'application est hautement disponible et accessible de façon transparente comme si l'utilisateur s'adressait à un service centralisé. Les gestionnaires d'applications sont indépendants. Par conséquent, le service d'ordonnancement fournit un service au mieux qui ne garantit pas un placement optimal des applications tel que pourrait le garantir un ordonnanceur centralisé mais qui a l'avantage de pouvoir être utilisé quelle que soit la taille de la grille et le nombre d'applications soumises.

En plus d'être tolérant aux défaillances, un service d'ordonnancement distribué permet de répartir la charge liée au calcul d'ordonnancement et ainsi de s'affranchir de l'utilisation de machines puissantes dédiées.

3.2.2 Large applicabilité du système

Nous proposons de concevoir un système unique capable de fédérer les ressources de n'importe quel type de grille et capable de supporter une large gamme d'applications utilisées dans le domaine de la simulation numérique.

Support de n'importe quel types de grille L'intergiciel que nous proposons est suffisamment générique pour supporter tous les types de grille grâce à deux caractéristiques.

Tout d'abord, les services distribués que nous proposons peuvent être utilisés dans un contexte où les nœuds sont volatils. Ainsi, nous pouvons envisager une utilisation de notre système dans une grille très dynamique de type *calcul pair-à-pair*. Par conséquent, notre système peut être utilisé avec des grilles moins dynamiques telles que les *fédérations de grappes*.

Ensuite, nous proposons un système capable d'utiliser deux types de nœuds qui sont les nœuds exploités en mode interactif comme une station de travail ou un calculateur sous Linux et les nœuds exploités avec un gestionnaire de traitement par lots comme une grappe exploitée avec Torque [35]. Ainsi, les applications soumises par les utilisateurs peuvent être indifféremment exécutées sur ces deux types de nœuds, et cela de façon transparente, ce qui permet d'envisager l'utilisation sur n'importe quel type de grille.

Support d'une large gamme d'applications Les services que nous proposons pour l'exécution d'applications distribuées sur une grille sont de niveau système. Ainsi notre proposition garantit la transparence nécessaire à l'exécution d'applications patrimoniales puisqu'elle ne requiert pas la modification des applications.

L'exécution d'applications distribuées sur une grille peut nécessiter plusieurs opérations selon le type d'application considéré.

Tout d'abord, il peut être nécessaire d'allouer plusieurs ressources simultanément pour l'exécution de tâches communicantes. Notre approche consiste à fournir un langage permettant aux utilisateurs de décrire simplement des groupes de tâches devant être exécutées simultanément. Pour les applications qui nécessitent de connaître la localisation des nœuds utilisés lors de l'exécution, nous proposons de fournir un mécanisme permettant de désigner les nœuds qui sont utilisés pour une exécution sans connaître leur localisation physique. Ce mécanisme peut être vu comme un service de DNS capable de modifier ses entrées dynamiquement en fonction de l'application considérée et des nœuds qu'elle utilise. Ainsi, l'utilisateur peut composer un script permettant de lancer une application en utilisant ces noms logiques pour désigner les nœuds qui seront utilisés lors de l'exécution sans qu'il ait à connaître la localisation réelle de ces nœuds. Typiquement, ces noms logiques peuvent être utilisés pour créer le `machinefile` utilisé par les applications fondées sur le paradigme MPI. Par ailleurs, pour garantir une exécution efficace des applications composées de tâches communicantes, nous proposons un mécanisme fondé sur l'utilisation de sondes réseau permettant d'évaluer la qualité des liens réseau entre les nœuds et de choisir au mieux des nœuds qui permettent aux tâches de communiquer efficacement.

D'autres types d'applications comme les *workflows* sont composés de tâches dont le démarrage de certaines est conditionné par la terminaison d'autres. De façon similaire au support pour l'exécution de tâches communicantes, le langage fourni aux utilisateurs pour décrire l'architecture des applications permet également d'exprimer des relations entre les tâches d'une application. Il est ainsi possible de définir que pour démarrer, une tâche a besoin d'un fichier produit par l'exécution d'une autre tâche.

Nous détaillons le langage de description des applications et les mécanismes associés dans le chapitre 5.

Notre système permet également d'exécuter des applications pilotées par un moteur externe de *workflow* comme nous le montrons avec le couplage de la plate-forme de simulation numérique SALOME [31] à notre intergiciel dans le chapitre 7.

3.2.3 Exécution simple, efficace et fiable des applications sur la grille

Notre approche permet de concilier trois caractéristiques pour l'exécution d'applications dans une grille de grande taille qui sont : la simplicité, l'efficacité et la fiabilité.

3.2.3.1 Simplicité

Utiliser un grand nombre de nœuds de calcul distribués et piloter des applications composées de nombreuses tâches telles que les applications de simulation numérique sont des opérations qui ne sont pas triviales pour des utilisateurs non spécialistes en informatique distribuée.

Pour simplifier l'utilisation d'une grille, nous proposons un intergiciel offrant aux utilisateurs une vue de type *système à image unique* et automatisant les opérations liées à l'exécution d'une application.

Système à image unique La principale source de complexité pour l'utilisation des nœuds d'une grille est liée au grand nombre de nœuds hétérogènes et volatils. Pour lever cette complexité d'utilisation, un intergiciel fédérant les ressources d'une grille doit masquer la distribution, l'hétérogénéité et la volatilité des ressources aux utilisateurs.

Notre approche consiste à offrir une vision de type SIU des ressources de la grille. En se connectant à n'importe quel nœud de la grille, un utilisateur possède une vision unique des nœuds.

Lorsqu'il soumet une application à la grille, cette dernière peut être exécutée sur n'importe quel nœud capable de l'exécuter sans que l'utilisateur n'ait à spécifier la localisation de ce nœud.

Après le démarrage de l'application, l'utilisateur n'a pas besoin de connaître la localisation du nœud utilisé car les opérations d'envoi des fichiers binaires et des fichiers d'entrée de l'application sur le nœud choisi pour l'exécution et les opérations de rapatriement des fichiers de sortie sont réalisées de façon transparente.

Concernant les opérations pouvant être réalisées au cours de l'exécution d'une application comme la demande du statut de l'exécution ou encore l'interruption de l'exécution, elles peuvent être initiées depuis n'importe quel nœud de la grille. De la même façon qu'un utilisateur a la possibilité d'agir sur l'exécution d'un processus s'exécutant sur une station de travail, il peut faire de même en étant connecté à n'importe quel nœud d'une grille avec les applications s'exécutant sur des nœuds distants dont il n'a pas besoin de connaître le nom.

Gestion automatique du cycle de vie des applications L'exécution d'une application nécessite la réalisation de plusieurs opérations entre la soumission et la récupération des résultats.

Pour simplifier l'exécution d'applications, nous proposons un intergiciel capable de prendre en charge le cycle de vie complet des applications et d'automatiser les opérations suivantes :

- l'analyse d'une requête de soumission ;
- l'initiation d'une découverte de ressources ;
- l'initiation d'une allocation de ressources ;
- l'initiation des transferts de fichiers d'entrée et de sortie ;
- la surveillance de l'application ;
- la réaction aux défaillances que peut subir l'application ;
- le nettoyage des ressources.

La gestion du cycle de vie des applications est détaillée dans le chapitre 5.

3.2.3.2 Efficacité

L'exécution efficace des applications repose sur deux caractéristiques liées à l'allocation des ressources qui sont le choix de ressources peu chargées et la prise en compte des contraintes de communication entre les tâches des applications.

Répartition des applications sur les ressources disponibles Nous proposons un système d'information permettant de maximiser les chances de trouver des ressources libres dans une grille. Couplé à un service d'allocation de ressource doté d'une politique choisissant les ressources les moins chargées, ce système d'information permet de maximiser l'allocation de ressources libres et ainsi de répartir au mieux les applications sur les nœuds de la grille. L'objectif est de limiter les risques d'exécution en concurrence qui peuvent dégrader les performances des applications. Nous détaillons les particularités du système d'information et les politiques d'allocation de ressources dans le chapitre 4.

Prise en compte des contraintes de communication des applications distribuées Certains types d'applications distribuées comme les applications parallèles ou les couplages de codes sont composés de tâches communicantes. Ces applications sont sensibles à la qualité du réseau d'interconnexion des nœuds utilisés pour l'exécution, en particulier pour ce qui concerne la latence.

Nous proposons un service d'allocation de ressources capable d'interpréter les spécifications d'une application fournies par l'utilisateur pour ce qui concerne les communications inter-tâches. Ce service permet de co-allouer des nœuds ayant une interconnexion réseau avec la latence la plus faible possible pour l'exécution efficace d'applications composées de tâches fortement communicantes. Ces mécanismes reposent sur l'utilisation de sondes réseau capables d'évaluer la latence entre les nœuds de la grille. Nous détaillons ces mécanismes dans le chapitre 4.

3.2.3.3 Fiabilité

Nous proposons de garantir la fiabilité de l'exécution des applications en utilisant des mécanismes de supervision et en permettant au gestionnaire de cycle de vie d'une application de réagir vis-à-vis des défaillances.

Supervision des nœuds et des tâches Nous proposons une infrastructure de supervision à deux niveaux. Un premier niveau de supervision permet de détecter les défaillances d'applications liées à la défaillance de nœuds de grille. Ces défaillances peuvent être la conséquence d'une défaillance matérielle ou d'une défaillance d'un lien réseau. Un second niveau de supervision permet de détecter les défaillances des tâches qui peuvent être liées au contexte d'exécution. Un exemple de défaillance liée au contexte d'exécution est la tentative d'écriture dans un espace disque temporaire plein. Nous détaillons les mécanismes de supervision dans le chapitre 5.

Gestion du cycle de vie prenant en compte les défaillances Pour réagir à une défaillance détectée par les mécanismes de supervision, nous proposons d'inclure au gestionnaire de cycle de vie d'application la capacité de ré-exécuter une partie ou l'ensemble des tâches défaillantes d'une application sur des nouvelles ressources. Cela permet de garantir que l'exécution d'une application atteindra toujours son terme en dépit de la défaillance des nœuds de la grille si suffisamment de ressources sont disponibles et si le code de l'application ne comporte pas d'erreur. Nous détaillons les mécanismes d'exécution fiable de notre système dans le chapitre 5.

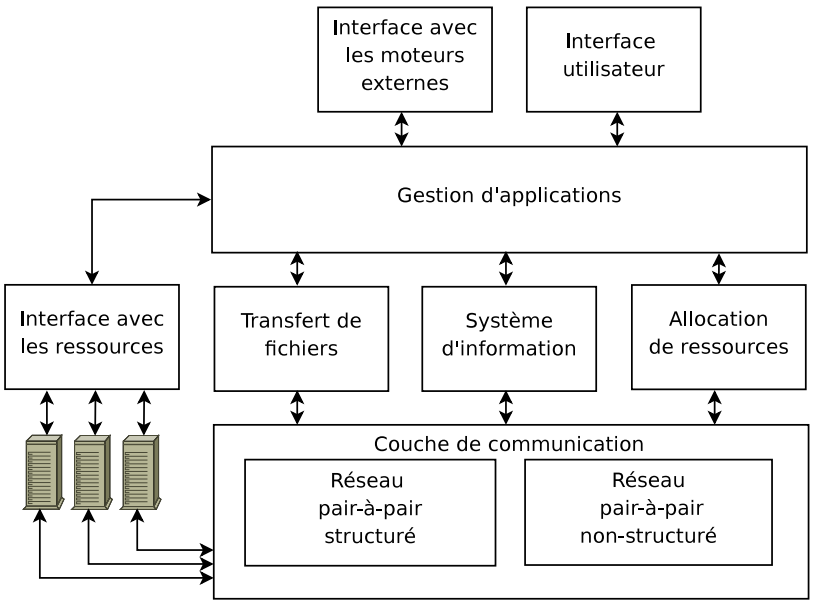


FIG. 3.6 – Architecture globale pour la gestion des ressources d'une grille

3.3 Architecture

La figure 3.6 présente l'architecture globale de l'intergiciel que nous proposons pour exécuter des applications distribuées sur une grille de grande taille composée de nœuds volatils et hétérogènes. Cette architecture a été entièrement mise en œuvre dans un prototype qui a servi de base à la validation de nos propositions présentées dans les chapitres 4, 5 et 7. Les résultats des évaluations sont présentés dans le chapitre 6.

Couche de communication Comme nous l'avons mentionné dans la partie 3.2.1.1, notre approche repose sur l'utilisation de réseaux pair-à-pair. Ces réseaux constituent la couche de communication du système. Elle permet aux services distribués qui composent le reste de l'architecture de communiquer entre eux de deux façons. La première est l'envoi de messages vers une clé de la THD (via le réseau pair-à-pair structuré) et la seconde est l'envoi vers des entités qui correspondent à certains critères (via le réseau pair-à-pair non structuré).

Interface avec les ressources Un de nos objectifs est de pouvoir utiliser des nœuds hétérogènes, notamment dans leur mode d'exploitation. Nous avons donc conçu une interface générique d'accès aux ressources qu'il suffit de spécialiser pour un type de gestionnaire de ressources donné.

L'interface d'accès aux ressources permet :

- la soumission d'une tâche ;
- la destruction d'une tâche ;
- la récupération de la charge du nœud ;
- le nettoyage des fichiers générés par l'exécution d'une tâche ;

- la création et le redémarrage d'un point de reprise d'une tâche ;
- la surveillance des processus d'une tâche.

Bien entendu, les interfaces spécialisées ne possèdent pas forcément une implémentation de toutes les fonctions de l'interface générique. C'est par exemple le cas pour les fonctions liées aux points de reprise qui sont supportées par un système comme Kerrighed [124] ou par un système GNU/Linux doté de BLCR [4], mais pas par le système GNU/Linux standard.

Transfert de fichiers Afin d'assurer la vision SIU dans le système Vigne, nous avons conçu un service de transfert de fichiers qui prend en charge tous les transferts de fichiers nécessaires pour l'exécution d'applications distribuées sur une grille. Sans proposer un système de fichiers distribué comme LegionFS [170] ou GFarm [163] puisque l'utilisateur doit spécifier les fichiers associés à une tâche, Vigne offre plus de transparence que Globus où l'utilisateur doit explicitement effectuer les déplacements de fichiers avec GridFTP [39].

Système d'information Le système d'information de Vigne permet de localiser des ressources dans le système qui correspondent aux critères requis par les applications des utilisateurs. Comme nous l'avons vu dans la partie 3.2.1.1, le système d'information de Vigne est fondé sur un réseau pair-à-pair non structuré. Ainsi, ce service est tolérant aux défaillances multiples et adapté à la grande échelle d'une grille. Le système d'information de Vigne est détaillé dans le chapitre 4.

Allocation de ressources Le service d'allocation de ressources est chargé d'allouer des ressources à une application. Les objectifs de l'allocation de ressources sont la répartition au mieux des applications sur les nœuds pour éviter les exécutions concurrentes. De plus, le service d'allocation de ressources permet aussi l'allocation de nœuds ayant une interconnexion réseau permettant l'exécution efficace d'applications composées de tâches communicantes. L'allocation de ressources est détaillée dans le chapitre 4.

Gestion d'application Le service de gestion d'application est chargé de gérer le cycle de vie complet d'une application et de traiter les défaillances que peut subir une application. Le service de gestion d'application est un point critique dans le système. Ainsi, ce service est distribué et ne repose sur aucune entité fiable. Grâce au réseau pair-à-pair structuré et aux travaux de Louis Rilling [144], ce service peut être rendu hautement disponible. La gestion d'application est détaillée dans le chapitre 5.

Interface avec les utilisateurs et les moteurs externes Le système Vigne fournit des interfaces avec le monde extérieur. Ces interfaces permettent d'exploiter le système pour effectuer par exemple une soumission d'application ou pour obtenir des informations sur une exécution. Pour renforcer la vision SIU, l'interface externe de Vigne est accessible depuis n'importe quel nœud de la grille et de façon asynchrone. Par exemple, une application peut être soumise sur un nœud et le suivi de son exécution ou la récupération des résultats peuvent être effectués depuis un autre nœud.

Il existe deux types d'interfaces externes. Tout d'abord, les utilisateurs peuvent soumettre leurs applications directement au système. Dans ce cas les utilisateurs fournissent une description des ressources nécessaires au fonctionnement de l'application (architecture processeur, quantité de mémoire vive nécessaire, système d'exploitation, etc) ainsi qu'une description de l'application (fichiers binaires, dépendances de synchronisation entre tâches, dépendances spatiales entre tâches, dépendances de précédence entre tâches, etc).

Il est aussi possible d'utiliser le système depuis un moteur externe qui est capable de gérer finement une application complexe où par exemple l'exécution de certaines tâches est conditionnée par le résultat d'autres tâches. Dans ce cas, un ensemble de fonctionnalités sont offertes, en plus de celles offertes à l'utilisateur, pour gérer par exemple le déplacement de fichiers de façon transparente entre les nœuds utilisés par les tâches ou encore pour synchroniser des tâches entre elles. Nous avons illustré cette fonctionnalité dans le chapitre 7 où nous avons utilisé la plate-forme SALOME comme un moteur externe.

3.4 Synthèse

L'exploitation des ressources d'une grille pour l'exécution d'applications distribuées est d'autant plus complexe que la taille de la grille est grande, que les ressources sont volatiles et que le nombre d'utilisateurs est grand.

Afin de fédérer les ressources réparties sur une grille de très grande taille, nous proposons tout d'abord de concevoir un intergiciel dont les services sont complètement distribués pour éliminer les risques de surcharge de certaines entités du réseau, de goulet d'étranglement et de point critique sensible aux défaillances. Notre approche repose sur l'utilisation de réseaux pair-à-pair qui possèdent des propriétés d'auto-réparation et d'auto-organisation pour supporter la volatilité des nœuds d'une grille. Ainsi, la défaillance d'un nœud quelconque dans la grille peut être tolérée sans empêcher le système de fonctionner correctement. Nous proposons également d'utiliser un ordonnanceur complètement distribué fondé sur l'idée d'utiliser un agent d'ordonnancement par application.

Nous proposons un intergiciel capable d'utiliser des ressources aussi volatiles que les ressources d'une grille de type *calcul pair-à-pair* et capable d'utiliser des ressources exploitées de façon interactive ou à l'aide d'un gestionnaire de traitement par lots. Par conséquent, notre proposition peut s'appliquer à la fédération des ressources de n'importe quel type de grille. Nous proposons un support transparent à l'exécution des applications distribuées composées de tâches communicantes et composées de tâches nécessitant le résultat d'autres tâches pour démarrer. Sans imposer la modification des applications, notre proposition permet d'exécuter une large gamme d'applications utilisées pour réaliser des simulations numériques, y compris les *workflows* gérés par un moteur externe.

Finalement, nous proposons des concepts permettant d'exécuter des applications distribuées de façon simple, efficace et fiable. La simplicité d'exécution est atteinte en offrant une vue de type SIU des nœuds qui masque leur distribution et leur volatilité. De plus, les opérations liées à l'exécution d'une application sur une grille sont complètement automatisées grâce à un gestionnaire de cycle de vie intégré au système. Nous proposons un service d'allocation de ressources dont l'objectif est de répartir l'ensemble des applications qui sont soumises sur

l'ensemble des ressources de la grille en tentant d'éviter les exécutions concurrentes pour une exécution efficace. De plus, nous proposons un mécanisme de co-allocation de ressources dont l'objectif est d'allouer des ressources situées sur des nœuds possédant une interconnexion réseau permettant d'exécuter efficacement les applications composées de tâches communicantes. Nous proposons un service de gestion d'application doté de mécanismes de supervision de nœuds et de tâches qui garanti que l'exécution d'une application puisse atteindre son terme en dépit des défaillances de nœuds ou des défaillances liées au contexte d'exécution qui ont une incidence sur le déroulement de l'exécution de l'application.

Nos contributions sont détaillées dans les chapitres 4, 5 et 7. À des fins d'évaluation, nous avons réalisé un prototype mettant en œuvre l'ensemble de nos propositions. Ce prototype ainsi que les résultats d'évaluation sont présentés dans les chapitres 6 et 7.

Chapitre 4

Le système d'information et l'allocation des ressources dans le système Vigne

Nous présentons dans ce chapitre deux services du système Vigne qui sont le système d'information et le service d'allocation de ressources. Dans un premier temps, nous étudions les objectifs et la complémentarité de ces deux services. Ensuite, nous détaillons chacun des deux services en présentant nos contributions.

4.1 Système d'information et allocation des ressources, deux services étroitement couplés

4.1.1 Objectifs du système d'information

Nous avons vu que dans une grille les nœuds peuvent être nombreux, hétérogènes et volatils. Le système ou les utilisateurs doivent avoir accès à une entité qui recense ces ressources pour les utiliser. Cette entité est appelée *système d'information*. Le système d'information d'une grille doit être capable de répondre à une requête de demande de ressources. Les requêtes sont habituellement constituées d'un ensemble de critères qui définissent les pré-requis pour l'exécution d'une application. Ces critères peuvent concerner des aspects matériels comme une architecture processeur, une quantité de mémoire requise ou encore une topologie réseau mais aussi des aspects logiciels comme un système d'exploitation ou encore la présence d'une librairie donnée.

Dans le cas d'un système d'information supportant la réservation des ressources, une requête de demande de ressources peut contenir des informations comme la durée prévue pour l'application soumise. Cela permet au système d'ordonnancer plus finement les applications. De plus, dans le cas d'un système d'information fondé sur un modèle économique comme celui de Nimrod/G [60], une requête de demande de ressources peut en plus contenir des critères comme l'heure souhaitée pour récupérer les résultats ou le prix maximal que l'utilisateur est prêt à payer. Toutefois les systèmes d'information utilisant des mécanismes de réservation de

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition xmlns="http://www.example.org/"
      xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
      xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <jsdl:JobDescription>
      <jsdl:Resources>
         <jsdl:ProcessorArchitecture>
            <jsdl:Exact>x86_64</jsdl:Exact>
         </jsdl:ProcessorArchitecture>
         <jsdl:OperatingSystem>
            <jsdl:Exact>FreeBSD</jsdl:Exact>
         </jsdl:OperatingSystem>
         <jsdl:IndividualPhysicalMemory>
            <jsdl:LowerBoundedRange>2097152.0</jsdl:LowerBoundedRange>
         </jsdl:IndividualPhysicalMemory>
         <jsdl:TotalCPUCount>
            <jsdl:Exact>1.0</jsdl:Exact>
         </jsdl:TotalCPUCount>
      </jsdl:Resources>
   </jsdl:JobDescription>
</jsdl:JobDefinition>
```

FIG. 4.1 – Exemple de requête de découverte de ressources au format JSDL

ressources ou ceux fondés sur un modèle économique sortent du cadre de cette étude.

La figure 4.1 montre un exemple de requête qui peut être soumise à un service d'information comprenant le format JSDL (*Job Submission Description Language*) proposé par l'OGF (*Open Grid Forum* [23]).

L'objectif du système d'information n'est pas forcément de trouver toutes les ressources de la grille qui correspondent à une requête, mais seulement un sous-ensemble. Nous avons vu dans la partie 2.3.1 que les systèmes d'information pouvaient être fondés sur différentes architectures. Selon l'approche, typiquement si l'architecture est distribuée, il pourrait être compliqué pour le système d'information de retourner une liste exhaustive des ressources qui correspondent à une requête donnée.

Le système d'information d'une grille peut maintenir un état très précis des ressources de la grille mais il peut aussi seulement garder la localisation des ressources sans état particulier. En fonction de cela, le service qui utilisera le système d'information ou l'utilisateur auront plus ou moins de travail à effectuer pour s'assurer que les ressources découvertes soient réellement disponibles.

4.1.2 Objectifs de l'allocation de ressources

L'allocation de ressources est l'action qui permet de choisir les ressources pour une application à partir des ressources qui ont été retournées par le système d'information et ceci, en fonction d'une politique donnée.

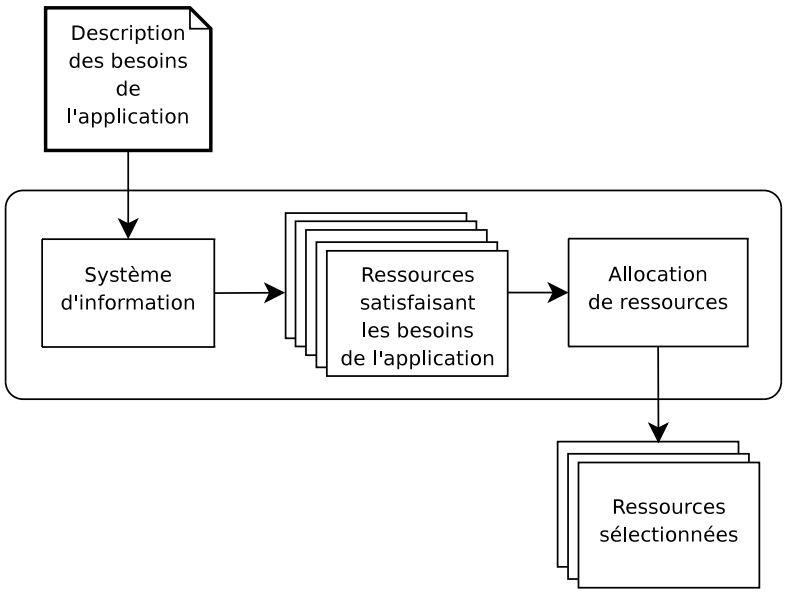


FIG. 4.2 – Lien entre le système d'information et le service d'allocation de ressources

Les politiques d'allocation de ressources peuvent avoir divers objectifs comme minimiser le temps d'exécution des applications [106], maximiser l'utilisation des ressources de la grille [166], économiser de l'énergie en n'utilisant qu'une partie des ressources [67, 112] ou encore utiliser un modèle économique [159, 173].

L'allocation de ressources peut donc être vue comme la phase d'ordonnancement dans le monde des grappes de calculateurs. En effet, dans le cas des grappes, les nœuds sont homogènes et statiques, il n'y a donc pas de phase de découverte de ressources pour l'ordonnancement d'applications.

De plus, l'allocation des ressources doit garantir à l'utilisateur que les ressources qu'il a demandées sont disponibles pour l'exécution de son application. En fonction du besoin des utilisateurs, les intergiciels de grille peuvent parfois fournir des fonctionnalités qui permettent soit de réserver exclusivement les ressources pour un utilisateur, soit de partager les ressources entre plusieurs utilisateurs.

4.1.3 Influence de la découverte des ressources sur l'allocation des ressources

Comme le montre la figure 4.2, le système d'information reçoit en entrée une description des besoins de l'application et fournit en sortie un certain nombre de ressources qui peuvent satisfaire les besoins de l'application. À partir de ces ressources le service d'allocation doit choisir les ressources qui seront réellement utilisées.

En fonction d'une politique de découverte de ressources donnée, l'application de politiques de sélection de ressources sera plus ou moins efficace. Plus le système d'information retourne de résultats, plus le choix du service d'allocation de ressources peut être effectué dans de bonnes conditions.

Dans un système de grille, l'objectif est souvent d'exécuter des applications le plus rapidement possible et ainsi d'allouer des ressources libres. Dans ce cas, il n'est pas souhaitable que le système d'information retourne au service d'allocation de ressources des ressources qui sont potentiellement encore occupées.

Selon les fonctionnalités offertes par le système d'information, comme par exemple la date de la dernière allocation ou l'état des ressources, la mise en place de politiques d'allocation de ressources pourra éventuellement être simplifiée.

4.2 Le système d'information de Vigne

Comme nous l'avons vu dans la partie 3.2.1.1, le système d'information de Vigne est fondé sur l'utilisation de réseaux logiques non structurés. Nous détaillons donc tout d'abord ce point. Ensuite, nous proposons une optimisation du protocole de marches aléatoires dont l'objectif est double. Cette optimisation permet de maximiser les chances de découvrir des ressources libres pour y exécuter des applications tout en limitant la bande passante utilisée par rapport à l'utilisation du concept d'inondation afin de limiter le coût de consultation du système d'information. Nous présentons également un mécanisme permettant d'augmenter les chances de trouver des ressources rares dans le réseau pair-à-pair non structuré, et donc d'améliorer la qualité du système d'information. Finalement, nous discutons de la pertinence de l'utilisation des réseaux logiques non-structurés pour la constitution du système d'information.

4.2.1 Utilisation de réseaux logiques non structurés

Souvent utilisés dans le contexte du partage de fichiers comme c'est le cas pour Gnutella [110], les réseaux pair-à-pair non structurés offrent des propriétés intéressantes pour la constitution d'un système d'information de grille. Ils permettent notamment de rechercher des ressources en spécifiant leurs caractéristiques. De plus, les réseaux pair-à-pair non structurés sont tout à fait adaptés à la recherche de ressources dans un système à grande échelle car aucune entité dans le système n'a besoin d'une connaissance globale de son environnement. Maintenir une connaissance globale des ressources du système est une tâche dont la complexité augmente avec l'échelle du système.

4.2.1.1 Principe de la recherche dans les réseaux non structurés

Nous avons vu dans la partie 3.2.1.1, que le moyen le plus simple d'effectuer des recherches dans un réseau non structuré est l'inondation de requêtes. Pour contrôler le nombre de requêtes il est possible de limiter la profondeur d'inondation avec un TTL. D'autres mécanismes peuvent toutefois être utilisés comme par exemple les marches aléatoires. Le concept de marche aléatoire a tout d'abord été étudié dans un contexte mathématique [37, 77]. Ensuite, ce concept a été repris dans le contexte des réseaux pair-à-pair [94, 99]. Dans ce cas, les marches aléatoires sont vues comme un raffinement du concept d'inondation. En effet, cela permet de limiter la bande passante utilisée pour une recherche. Le fonctionnement est le suivant. Au lieu d'envoyer une requête à tous ses voisins dans le réseau non-structuré, un nœud envoie la requête uniquement à un de ses voisins, choisi aléatoirement. De plus, lorsqu'un nœud reçoit une requête, il

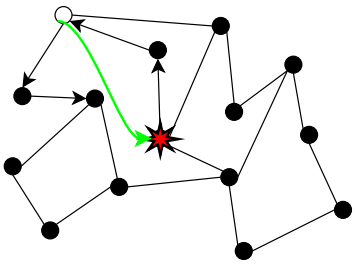


FIG. 4.3 – Première phase d'une découverte de ressources par marches aléatoires

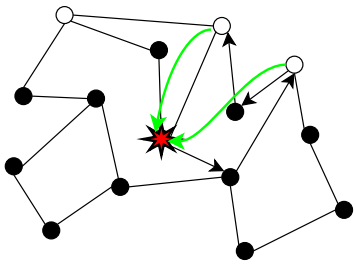


FIG. 4.4 – Seconde phase d'une découverte de ressources par marches aléatoires

l'évalue et répond à l'émetteur. Il fait également suivre cette requête à un de ses voisins. La profondeur d'une marche aléatoire est elle aussi limitée par un TTL. À l'issue d'une marche aléatoire, l'émetteur de la requête peut juger qu'il n'a pas reçu suffisamment de réponses. Dans ce cas, il peut initier d'autres marches aléatoires.

La figure 4.3 illustre un exemple où le nœud marqué d'une étoile initie une découverte de ressources en utilisant des marches aléatoires avec un TTL de 4. Nous supposons que ce nœud recherche trois ressources d'un type donné. À l'issue d'une première marche aléatoire, il a seulement reçu une réponse, du nœud coloré en blanc. Il doit donc initier une autre marche aléatoire pour obtenir plus de réponses. C'est ce qu'illustre la figure 4.4. Cette fois deux autres nœuds répondent à la requête, ce qui fait trois réponses en tout. Aucune autre marche aléatoire n'est donc initiée. Comparativement au protocole d'inondation classique, il est aisé de voir que le nombre de messages est fortement réduit car la découverte de ressources s'arrête dès que suffisamment de réponses ont été obtenues.

Dans [99], le concept de marche aléatoire a lui aussi été raffiné. Plutôt que d'envoyer les requêtes à un voisin choisi au hasard, diverses politiques sont définies. Par exemple, il est possible de choisir le voisin qui a répondu précédemment au plus grand nombre de requêtes. Ou encore, il est possible de choisir le voisin qui a répondu le plus souvent à une requête similaire. Ces politiques nécessitent la mise en place d'un cache sur chaque nœud afin de mémoriser les informations sur le nombre de requêtes répondues ou sur le type des requêtes répondues.

4.2.1.2 Composition du réseau pair-à-pair non-structuré

Afin de constituer le graphe de connexion du réseau pair-à-pair non structuré, chaque nœud doit se connecter à un certain nombre de voisins. Pour réaliser cette composition, nous avons utilisé le protocole Scamp [92]. L'idée de Scamp est de permettre un degré de connexion uniforme de chaque nœud qui est de l'ordre de $(c+1)log(n)$ si n est le nombre de nœuds dans le réseau et c le nombre souhaité de défaillances simultanées supportées. Le protocole Scamp fonctionne de la façon suivante. Pour se joindre au réseau, un nœud envoie une demande d'inscription à n'importe quel nœud du réseau. Ce nœud est appelé nœud d'amorçage (ou *bootstrap* en anglais). Comme le montre l'algorithme 1, à la réception de cette requête d'inscription, le nœud d'amorçage l'envoie à tous ses voisins (cf. lignes 2 à 4). D'autres envois peuvent être effectués si le paramètre c permettant de supporter plusieurs défaillances simultanées est supérieur à 0. Par la suite, un nœud qui reçoit cette requête (voir l'algorithme 2) vérifie s'il connaît

Algorithme 1 Gestion d'une demande d'inscription avec le protocole Scamp

```
1: procedure RECEPTIONDEMANDEINSCRIPTION(demandeur, c)
2:     for (i ← 0, i < taille(listeVoisins), i++) do
3:         Envoyer(listeVoisins[i], demandeur, 'msg_demande_inscription')
4:     end for each
5:     for (j ← 0, j < c, j++) do
6:         hasard = ChoixHasard(0, taille(listeVoisins) − 1)
7:         Envoyer(listeVoisins[hasard], demandeur, 'msg_demande_inscription')
8:     end for each
9: end procedure
```

Algorithme 2 Réception d'une requête d'inscription avec le protocole Scamp

```
1: procedure RECEPTIONREQUETEINSCRIPTION(demandeur)
2:     garde ← ChoixHasard(0, 1)
3:     garde ← EntierLePlusProche((taille(listeVoisins) + 1) × garde)
4:     if (garde = 0) ∧ (demandeur ∉ listeVoisins) then
5:         Ajouter(demandeur, listeVoisins)
6:     else
7:         hasard ← ChoixHasard(0, taille(listeVoisins) − 1)
8:         Envoyer(listeVoisins[hasard], demandeur, 'msg_demande_inscription')
9:     end if
10: end procedure
```

déjà le nœud voulant se joindre au réseau. S'il ne le connaît pas, il l'ajoute dans sa liste de voisins avec une probabilité de $\frac{1}{(1+nombre\_de\_voisins)}$. Dans le cas contraire, la requête est envoyée à un nœud choisi au hasard dans la liste des voisins.

Scamp propose aussi des mécanismes pour équilibrer le réseau dans le cas où des nœuds quittent le réseau. Il faut en effet garantir que le nombre de voisins d'un nœud soit toujours suffisant sinon la découverte de ressources devient inefficace. Nous n'avons toutefois pas utilisé ces mécanismes, car comme nous allons le voir dans le paragraphe 4.2.1.3, il est possible d'adopter une méthode plus simple avec le système Vigne.

Nous avons vu dans la partie 3.2.1.1 que le système Vigne utilisait également un réseau pair-à-pair structuré pour assurer les communications nécessaires des différents services du système. Le processus d'amorçage est simplifié grâce au réseau structuré puisqu'il suffit d'envoyer la requête d'inscription à un nœud présent dans la table de routage du réseau structuré. Cette table de routage est automatiquement mise à jour grâce aux protocoles Pastry [150] et Bamboo [141], ce qui garantit que le nombre de voisins reste suffisant en dépit de la volatilité des nœuds. Ceci est détaillé dans le paragraphe 4.2.1.3.

4.2.1.3 Maintenance du réseau logique non-structuré

Afin de maintenir un nombre suffisant de voisins dans la vue de chaque nœud, le réseau logique non-structuré doit être continuellement mis-à-jour pour faire face à la volatilité des nœuds de la grille.

Lorsque des ressources sont ajoutées à la grille, le protocole de composition fondé sur Scamp [92] garantit que les vues d'un nombre suffisant de nœuds sont mises à jour.

Pour gérer le départ des nœuds, nous utilisons un mécanisme qui contrôle régulièrement

si les nœuds présents dans la vue d'un nœud sont toujours atteignables dans la grille. Ce mécanisme repose sur l'utilisation, sur chaque nœud, d'un temporisateur qui permet d'exécuter à intervalles réguliers l'algorithme 3.

Algorithme 3 Vérification du voisinage d'un nœud dans le réseau non-structuré

```
 1: procedure VERIFICATIONVOISINAGE
 2:     hasard ← ChoixHasard(0, taille(listeVoisins) − 1)
 3:     voisin ← listeVoisins(hasard)
 4:     maintenant ← ObtenirHeure()
 5:     if (maintenant − DerniereCommunication(voisin)) > δ then
 6:         retour ← EnvoyertPingBloquant(voisin)
 7:         if retour = ECHEC then
 8:             listeVoisins ← listeVoisins − {voisin}
 9:             if taille(listeVoisins) < β then
10:                 DemandeInscriptionScamp()
11:             end if
12:         end if
13:     end if
14: end procedure
```

Afin de ne pas consommer trop de bande passante pour la maintenance des vues des nœuds dans le réseau non-structuré, un seul nœud est contrôlé lorsque l'algorithme est exécuté (cf. lignes 2 et 3). Si une communication vers ce nœud a eu lieu moins de δ secondes auparavant (ligne 5), il n'est pas nécessaire d'effectuer d'autre test. Dans le cas contraire, un ping est envoyé de façon bloquante vers le nœud (ligne 6). Si le pong n'est pas reçu (ligne 7) alors ce nœud est retiré du voisinage (ligne 8). De plus, si la taille de la vue est inférieure à β, alors une demande d'inscription dans le réseau est envoyée afin de trouver de nouveaux voisins.

Lorsque le protocole de maintenance du réseau structuré détecte qu'un nœud n'est plus dans le réseau, l'algorithme 4 est exécuté.

Algorithme 4 Notification au protocole de maintenance du réseau structuré de la détection d'une absence dans le réseau structuré

```
1: procedure NOTIFICATIONABSENCERESEAUSTRUCTURE(noeudAbsent)
2:     if (noeudAbsent ∈ listeVoisin then
3:         listeVoisins ← listeVoisins − {noeudAbsent}
4:         if taille(listeVoisins) < β then
5:             DemandeInscriptionScamp()
6:         end if
7:     end if
8: end procedure
```

Dans ce cas, si le nœud dont l'absence a été détectée est présent dans le voisinage du point de vue du réseau non-structuré, alors il est retiré du voisinage. De plus, une vérification est effectuée pour s'assurer qu'il y a plus de β nœuds dans le voisinage. Si ce n'est pas le cas, une demande d'inscription dans le réseau est envoyée afin de trouver de nouveaux voisins.

De même que pour l'amorçage, il y a une coopération entre le réseau logique structuré et le réseau logique non structuré pour ce qui concerne la détection des défaillances. Ainsi, lorsqu'un voisin de la vue du réseau structuré est détecté comme défaillant, son statut peut être marqué

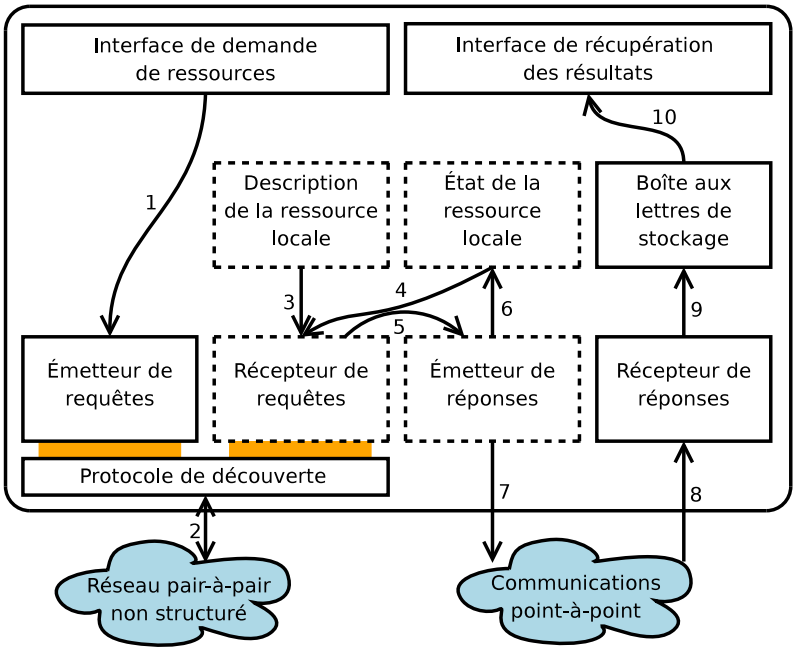


FIG. 4.5 – Architecture du système d'information de Vigne

comme défaillant s'il est présent dans la vue du réseau non-structuré, et réciproquement.

4.2.1.4 Architecture du service d'information

La figure 4.5 présente l'architecture du système d'information de Vigne. Cette architecture logicielle est présente sur chaque nœud de la grille fédéré par Vigne. Dans cette architecture, les boîtes dessinées en trait plein correspondent aux fonctionnalités qui sont utilisées sur le nœud initiateur d'une découverte de ressources et celles en trait pointillé correspondent aux fonctionnalités utilisées sur les nœuds qui reçoivent les requêtes.

L'objectif d'un système d'information étant de retourner des ressources capables d'exécuter les applications des utilisateurs, ce service possède naturellement une interface pour soumettre une demande de ressources et une autre pour récupérer les résultats de la demande.

Lorsqu'une demande est effectuée, une requête est créée par le système d'information et transmise à l'émetteur de requêtes (flèche 1). Selon le protocole de découverte de ressources choisi, la requête est envoyée dans le réseau pair-à-pair non structuré en utilisant une inondation, des marches aléatoires ou d'autres variantes (flèche 2).

À la réception d'une requête (flèche 2), un nœud vérifie si ses ressources locales peuvent satisfaire les besoins spécifiés dans la requête (flèche 3) et si suffisamment de ressources sont libres pour l'exécution de la tâche (flèche 4). Si les conditions sont vérifiées, le récepteur de requête transmet une demande à l'émetteur de réponses (flèche 5). Lorsqu'un nœud répond à une requête de découverte de ressources, il n'est pas certain que ce nœud soit choisi par l'allocateur de ressources. Toutefois, s'il répond à une autre requête avant de savoir s'il a été choisi par l'allocateur pour la première, il existe une probabilité non nulle que ce nœud soit

choisi pour exécuter les deux tâches et qu'il se retrouve surchargé. Pour éviter cela, un nœud ne doit pas répondre à une requête de découverte de ressources tant qu'il n'a pas été choisi par l'allocateur de ressources ou qu'un délai correspondant au délai maximum d'une découverte de ressources ne s'est pas écoulé. Ainsi, après avoir répondu de façon positive à l'initiateur d'une requête de découverte de ressources, l'émetteur de réponses verrouille temporairement le nœud en positionnant son état comme `REPONSE_IMPOSSIBLE` (flèche 6). L'état du nœud repassera à `REPONSE_POSSIBLE` s'il est alloué ou si le délai de garde est passé. L'envoi de la réponse est effectué de façon directe en utilisant une communication point-à-point (flèche 7).

Lorsque l'initiateur de la découverte de ressources reçoit une réponse (flèche 8), il la stocke dans une boîte aux lettres (flèche 9). Étant donné que chaque découverte de ressources peut être identifiée de façon unique, toutes les réponses d'une découverte de ressources peuvent être récupérées ultérieurement dans la boîte aux lettres via l'interface de récupération des résultats (flèche 10). De plus, l'identification unique des découvertes de ressources permet d'initier plusieurs découvertes depuis un même nœud.

Lorsque les résultats d'une découverte de ressources sont récupérés, les réponses sont supprimées de la boîte aux lettres et les éventuelles requêtes qui seraient reçues ultérieurement pour cette découverte de ressources sont ignorées.

Doté du mécanisme de boîte aux lettres, le fonctionnement du système d'information est asynchrone. Cela permet de supporter simplement la défaillance des nœuds puisqu'aucune réponse n'est attendue si un nœud ne peut pas satisfaire la requête. De plus, l'asynchronisme permet de soumettre une demande de découverte de ressources et de ne pas rester bloquer pendant l'attente des résultats.

4.2.2 Optimisation des marches aléatoires pour l'allocation des ressources

Nous avons vu dans la partie 4.1.3 que la découverte et l'allocation des ressources sont finement liées. Dans le contexte d'utilisation des grilles composées de ressources hétérogènes et volatiles, l'objectif est d'allouer des ressources libres à des applications. Dans ce sens, nous avons optimisé le protocole de marches aléatoires afin que la découverte de ressources retourne les résultats les plus adaptés à une allocation ultérieure, c'est-à-dire, des ressource potentiellement libres. Nous montrons dans un premier temps les différentes optimisations réalisées et nous présentons finalement le protocole dans sa globalité.

4.2.2.1 Utilisation de caches

L'utilisation de marches aléatoires permet de réduire la bande passante consommée pour une découverte de ressources. Il est toutefois possible de réduire encore cette consommation en utilisant des caches. Un cache est présent sur chaque nœud et contient les informations relatives à un certain nombre de ressources. Les informations mémorisées dans le cache pour un nœud donné sont : l'adresse logique du nœud enregistré, les ressources dont il dispose (matérielles et logicielles), la date à laquelle le nœud a été enregistré dans le cache et la date de sa dernière allocation si elle est connue.

Les données sont enregistrées dans le cache à l'issue de chaque découverte de ressources, lorsque le nœud émetteur de la découverte de ressources reçoit des réponses.

Ainsi, lorsqu'un nœud effectue une requête de découverte de ressources, le cache est consulté et si des ressources présentes dans le cache peuvent répondre à la demande, elles sont contactées en priorité. Ensuite, si le nombre de réponses reçu est insuffisant, la découverte de ressources est complétée par des marches aléatoires.

4.2.2.2 Utilisation d'une politique de gestion de cache favorisant la découverte de ressources libres

L'utilisation d'un cache permet d'économiser de la bande passante mais pour que ce cache ait un intérêt dans notre contexte d'utilisation, il faut lui adjoindre une politique de gestion efficace.

Un cache n'est utile que si les données qu'il possède sont suffisamment à jour. Intuitivement, nous pouvons nous dire que les informations dans le cache ne sont utiles que si elles correspondent à des ressources libres. Par conséquent, notre politique de gestion de cache favorise les nœuds dont les ressources n'ont pas été récemment allouées. Reprenons cela dans le contexte de la découverte de ressources dans Vigne. Nous avons vu dans la partie 4.2.1.4 que les réponses à une découverte de ressources étaient réceptionnées par le système d'information à l'aide du récepteur de réponses. Avec l'utilisation d'un cache, ces réponses réceptionnées sont ajoutées au cache si elles n'y sont pas déjà présentes. Par la suite, lorsque le service d'allocation de ressources choisit effectivement les ressources à utiliser parmi celles découvertes, il marque ces ressources comme allouées.

L'application de la politique de gestion cache intervient pour deux opérations qui sont : la consultation du cache et l'éviction du cache.

Lorsque qu'un nœud effectue une requête de découverte de ressources et que le cache contient plus de nœuds pouvant répondre à la requête que ce qui a été demandé, la politique de gestion de cache retourne en priorité les nœuds n'ayant jamais été alloués ou ceux dont la date d'allocation est la plus ancienne.

Lorsque le cache est plein et qu'un nouveau nœud doit y être ajouté, le nœud dont la date d'allocation est la plus récente est évincé. Si la date d'allocation ne permet pas de faire la différence entre les nœuds présents dans le cache, c'est l'entrée la plus ancienne qui est évincée et dans le cas où la date d'insertion ne permet pas non plus de faire la différence, une entrée est supprimée au hasard.

4.2.2.3 Diffusion partielle des informations découvertes

Dans le système Vigne, les découvertes de ressources peuvent être initiées depuis n'importe quel nœud de la grille et ceci, de façon aléatoire. Pour cette raison, la probabilité que deux requêtes de découverte de ressources soient initiées depuis le même nœud est relativement faible, surtout si le nombre de nœuds dans la grille est grand.

Étant donné que le cache d'un nœud est rempli à l'issue d'une découverte de ressources effectuée depuis ce nœud, il faut attendre un certain temps pour profiter des données insérées dans le cache, ce qui n'est pas souhaitable car les informations pourraient être obsolètes.

Afin d'améliorer la vitesse de remplissage des caches, nous avons ajouté un mécanisme de diffusion des informations découvertes. Ainsi, à l'issue d'une découverte de ressources, les réponses reçues, sauf celles correspondant aux ressources effectivement allouées, sont diffusées dans le voisinage du nœud initiateur de la découverte dans le réseau pair-à-pair non structuré.

Avec ce mécanisme, les caches se remplissent beaucoup plus vite, ce qui augmente considérablement leur intérêt. Cela pose toutefois un problème. Si un nœud ayant reçu une information diffusée de cette façon alloue la ressource concernée, le voisinage du nœud ayant diffusé l'information possède dans son cache une ressource récemment allouée sans le savoir, ce qui n'est pas très utile pour une allocation ultérieure. Pour cela, lorsqu'un nœud alloue une ressource dont les informations sont présentes dans son cache, il vérifie si ces informations proviennent d'une découverte de ressources ou d'une diffusion. Dans ce dernier cas, il prévient l'auteur de la diffusion de l'allocation de la ressource. Sur réception de ce type de message, le nœud qui avait effectué la diffusion d'information diffuse un message pour avertir ses voisins de l'allocation de la ressource concernée afin qu'ils puissent mettre à jour leur cache. Cela permet de garder les informations diffusées à jour dans les caches.

4.2.2.4 Algorithme complet du protocole de marches aléatoires optimisé

Nous présentons maintenant dans sa globalité notre version optimisée du protocole de marches aléatoires.

Notre protocole, comme celui des marches aléatoires, permet à partir d'une description de ressources de retourner, si possible, un certain nombre de ressources qui peuvent satisfaire la demande. Le nombre de ressources demandé peut être fixé par le système et doit au moins être égal au nombre de ressources requises pour l'exécution de l'application.

Algorithme 5 Découverte et allocation de ressources

```
1: procedure DECOUVERTEETALLOCATIONDERESSOURCES(caracteristiques, nbResultats)
2:     reponses ← MarchesAleatoiresOptimisees(caracteristiques, nbResultats)
3:     ressourceChoisie ← SelectionRessource(reponses)
4:     DiffusionInformations(ressourceChoisie, reponses)
5: end procedure
```

Ce protocole s'exécute en deux phases. Comme le montre l'algorithme 5, la première phase consiste à découvrir les ressources (cf. ligne 2) et la seconde phase consiste à diffuser les informations découvertes (cf. ligne 4). Entre ces deux phases s'intercale l'allocation des ressources. Pour simplifier l'algorithme, nous considérons ici qu'une seule ressource est choisie par l'allocateur de ressources.

La première partie du protocole qui consiste à découvrir des ressources est découpée en trois phases, comme le montre l'algorithme 6. La première phase (cf. lignes 3 à 6) consiste à rechercher dans le cache local les ressources qui correspondent aux caractéristiques requises. Si des ressources sont trouvées dans le cache, alors elles sont contactées directement (cf. ligne 5). Ensuite, le protocole attend les réponses des nœuds contactés directement (cf. lignes 7 à 16) et initie en parallèle des marches aléatoires tant que le nombre de réponses n'est pas suffisant (cf. lignes 13 à 15). Lorsqu'une réponse est reçue, elle est ajoutée dans le cache local si elle n'y est pas déjà présente. L'algorithme 7 est utilisé pour ajouter une ressource dans le cache

Algorithme 6 Marche aléatoire optimisée

```
1: function MARCHESALEATOIRESOPTIMISEES(caracteristiques, nbResultats)
2:     reponses ← ∅
3:     infosDuCache ← ChercherInfosDansCache(caracteristiques, nbResultats)
4:     if infosDuCache = ∅ then
5:         MarchesAleatoires(nbResultats, caracteristiques)
6:     else
7:         for each noeud ∈ infosDuCache do
8:             DemandeInformations(noeud, caracteristiques)
9:         end for each
10:    end if
11:    while (|reponses| < nbResultats) ∧ ¬(FinDuTempsDeDecouverte()) do
12:        reponse ← AttendreReponse()
13:        reponses ← reponses ∪ {reponse}
14:        if ¬(reponse ∈ cache) then
15:            AjoutRessourceDansCache(reponse)
16:        end if
17:        if reponses < nbResultats then
18:            MarchesAleatoires(nbResultats − reponses, caracteristiques)
19:        end if
20:    end while
21:    return reponses
22: end function
```

Algorithme 7 Ajout des informations d'une ressource dans le cache

```
1: procedure AJOUTRESSOURCEDANSCACHE(ressource)
2:     if |cache| ≥ cacheSize then
3:         nbSuppr ← SupprimerLaRessourceLaPlusRecemmentAllouee()
4:         if nbSuppr = 0 then
5:             nbSuppr ← SupprimerLaPlusVieilleRessourceAjoutee()
6:             if nbSuppr = 0 then
7:                 SupprimerUneRessourceAuHasard()
8:             end if
9:         end if
10:    end if
11:    cache ← cache ∪ ressource
12: end procedure
```

et gérer le cas où le cache est plein. Nous pouvons voir que la priorité de suppression d'une ressource dans le cache concerne tout d'abord la ressource la plus récemment allouée, puis celle ajoutée en premier dans le cache. Si ces deux priorités ne sont pas discriminantes, une ressource choisie au hasard est supprimée.

Algorithme 8 Réception d'une requête de découverte de ressources sur un nœud

```
1: procedure RECEPTIONREQUETEDECOUVERTERESSOURCES(caracteristiques, demandeur)
2:     if (ComparerAvecRessourceLocale(caracteristiques) ∧ RessourceLocaleEstLibre(caracteristiques) then
3:         ChangerEtatRessourceLocale(REPONSE_IMPOSSIBLE)
4:         EnvoieReponse(demandeur, caracteristiquesLocales, charge)
5:     end if
6: end procedure
```

L'algorithme 8 présente le traitement effectué lorsqu'un nœud reçoit une requête de découverte de ressource. Tout d'abord, le nœud vérifie s'il possède les caractéristiques demandées et si toutes ses ressources ne sont pas allouées à une autre application (cf. ligne 2). Si les conditions sont satisfaites, le nœud verrouille ses ressources pour empêcher temporairement la réponse à d'autres requêtes (cf. ligne 3). Ces ressources sont déverrouillées au bout d'un certain temps, si le nœud n'est pas choisi par l'allocateur de ressources. Finalement, une réponse est envoyée à l'initiateur de la découverte de ressources (cf. ligne 4). Dans cette réponse, le nœud envoie ses caractéristiques propres et sa charge actuelle. Dans le cas où le nœud a été contacté directement, c'est-à-dire s'il était présent dans le cache du nœud initiateur de la découverte de ressources, l'envoi des caractéristiques peut être optionnel. Toutefois, si l'on suppose que les nœuds sont très volatils et qu'ils subissent fréquemment des mises à jour matérielles ou logicielles, il est préférable de retourner cette information afin que l'information soit mise à jour dans le cache.

Algorithme 9 Diffusion des informations découvertes

```
1: procedure DIFFUSIONINFORMATIONSDECOUVERTES(resourceChoisie, reponses)
2:     reponsesADiffuser ← ∅
3:     for each reponse ∈ reponses do
4:         if reponse ≠ ressourceChoisie then
5:             reponsesADiffuser ← reponsesADiffuser ∪ reponse
6:         else
7:             provenanceReponse ← RecupererLaProvenanceDeLaRessource(reponse)
8:             if provenanceReponse ≠ "Recherche locale" then
9:                 AvertirAllocation(provenanceReponse, reponse)
10:            end if
11:        end if
12:    end for each
13:    for each voisin ∈ listeVoisins do
14:        EnvoieInformationsDecouvertes(voisin, reponsesADiffuser)
15:    end for each
16: end procedure
```

La deuxième phase du protocole optimisé, se déroulant après l'allocation des ressources, consiste à diffuser les informations découvertes afin de mettre à jour les caches des nœuds voisins d'un nœud ayant initié une découverte de ressources. Comme le montre l'algorithme 9,

les informations découvertes sont diffusées à tous les voisins dans le réseau pair-à-pair non structuré du nœud initiateur de la découverte de ressources. Dans le cas où le nœud découvert a été alloué, deux traitements sont possibles :

- soit ce nœud avait été enregistré dans le cache à l'issue d'une réception de diffusion. Dans ce cas, un message est envoyé au nœud qui avait initié cette diffusion pour lui demander d'avertir ses voisins de l'allocation de la ressource (cf. lignes 8 à 10) ;
- soit ce nœud provient d'une découverte de ressources par marche aléatoire ou alors d'une présence dans le cache non liée à une diffusion préalable, dans ce cas rien n'est fait puisqu'il n'est pas intéressant de faire connaître une ressource qui vient d'être allouée.

Algorithme 10 Réception d'une information diffusée

```
1: procedure RECEPTIONINFORMATIONDIFFUSEE(infosRessource)
2:     if infosRessource ∉ cache then
3:         AjoutRessourceDansCache(infosRessource)
4:     else
5:         MettreCacheAJour(infosRessources)
6:     end if
7: end procedure
```

La réception d'une information diffusée par un nœud est spécifiée dans l'algorithme 10. Si l'information sur la ressource n'est pas présente dans le cache, alors elle y est ajoutée, sinon elle est mise à jour.

4.2.3 Trouver des ressources rares dans le système d'information

Dans les grilles hétérogènes, divers types de ressources peuvent être présents y compris des ressources particulièrement rares comme par exemple un super-calculateur parallèle, un calculateur ayant une architecture processeur peu commune ou encore un calculateur possédant une quantité de mémoire largement supérieure à la norme. L'utilisation d'un système d'information fondé sur une architecture distribuée peut poser le problème de découverte de ressources rares dans le système. En effet, lorsqu'une inondation est lancée dans le réseau pair-à-pair non structuré, le paramètre TTL peut empêcher la couverture complète du graphe des nœuds. De même avec les marches aléatoires, afin de ne pas attendre indéfiniment le résultat d'une découverte de ressources, la durée maximale de la découverte de ressources est bornée par un paramètre. Il est donc tout à fait probable qu'une ressource très rare ne soit pas découverte.

Le problème de trouver une ressource rare se divise en deux sous-problèmes qui sont la détermination de la rareté d'une ressource et l'augmentation de sa popularité.

4.2.3.1 Déterminer la rareté d'une ressource

S'il est simple de déterminer la rareté d'une ressource dans un système d'information centralisé puisque cela se résume à la consultation d'une liste ou d'une base de données, il en va tout autrement pour un système d'information distribué.

Une approche simple pour déterminer la rareté d'une ressource consiste à demander cette information au fournisseur de la ressource lorsqu'il l'ajoute dans la grille. Compte-tenu de la

grande échelle potentielle, il est peu probable qu'un fournisseur de ressource connaisse l'ensemble des ressources mises à disposition dans la grille. Un fournisseur ne peut donc pas évaluer simplement la rareté des ressources qu'il fournit. Ainsi, nous souhaitons que cette opération soit réalisée automatiquement.

Nous proposons de faire une évaluation statistique de la rareté d'une ressource. Nous partons des hypothèses suivantes :

1. un type de ressource que l'on ne trouve pas facilement est considéré comme rare ;
2. une ressource qui peut fréquemment répondre de façon favorable aux requêtes de découverte de ressources (sans tenir compte de son état d'occupation) n'est pas considérée comme rare.

L'idée est que les nœuds qui composent la grille interrogent de façon aléatoire dans le réseau un ensemble de nœuds et comparent leurs ressources locales avec celles des nœuds interrogés. Si dans l'échantillon de nœuds interrogés, les ressources locales sont différentes des ressources des autres nœuds, la ressource locale peut être considérée comme rare. Compte-tenu de la volatilité des ressources composant la grille, ce traitement doit être effectué régulièrement. Toutefois, si tous les nœuds du système effectuent de telles interrogations, une bande passante importante risque d'être consommée. D'après la seconde hypothèse que nous avons donnée, un certain nombre de nœuds peuvent être considérés comme possédant des ressources non rares dès lors qu'ils ont été capables de donner de nombreuses réponses positives aux requêtes de découverte de ressources. Il n'est donc pas nécessaire que ces ressources effectuent les interrogations statistiques, ce qui permet d'économiser de la bande passante.

L'algorithme 11 décrit comment un nœud détermine la rareté des ressources qu'il héberge. Tout d'abord, le nœud évalue s'il a besoin ou non d'effectuer cette interrogation (cf. ligne 4). Dans l'hypothèse où il en a besoin, il effectue des marches aléatoires pour trouver α ressources de n'importe quel type (cf. ligne 7). α doit être une fraction de la taille potentielle de la grille, typiquement 5 ou 10 %. La taille de la grille peut être déterminée approximativement ou encore avec des techniques comme [119] qui utilisent les marches aléatoires pour estimer la taille d'un réseau pair-à-pair non structuré et qui possèdent l'avantage de fournir une évaluation précise de la taille de la grille lorsque cette dernière varie dynamiquement. Une fois l'attente des réponses terminées (cf. lignes 8 à 11), le nœud évalue parmi les réponses le nombre de ressources qui sont similaires à celles qu'il propose (cf. lignes 12 à 16). Finalement, si la proportion de ressources similaires est inférieure à une valeur β (typiquement 2 à 5 %), alors le nœud doit se faire connaître (cf. lignes 18). Si cette proportion est supérieure à β alors le coefficient de rareté est diminué (cf.ligne. 20). Ce traitement est effectué toutes les γ minutes. γ doit être adapté à la volatilité des ressources.

L'algorithme 8 doit être légèrement adapté pour que le coefficient de rareté des nœuds puisse être mis à jour. Le résultat de l'adaptation est montré dans l'algorithme 12. Il faut simplement diminuer le coefficient de rareté lorsque le nœud peut répondre à une requête de découverte de ressources.

4.2.3.2 Faire connaître une ressource rare

Même si un nœud détermine qu'il héberge des ressources rares, cela ne suffit pas pour que cette ressource puisse être découverte plus simplement compte-tenu de l'architecture distribuée

Algorithme 11 Évaluation réalisée par un nœud pour savoir si ses ressources sont rares

```
1: procedure EVALUATIONRARETERESSOURCE
2:     while vrai do
3:         coefficientRarete ← coefficientRarete × 1,1
4:         if coefficientRarete > 1 then
5:             reponses ← ∅
6:             nbRessourcesSimilaires ← 0
7:             MarchesAleatoires(α,"toute ressource")
8:             while (|reponses| < α) ∧ ¬(FinDuTempsDInterrogation()) do
9:                 reponse ← AttendreReponse()
10:                reponses ← reponses ∪ {reponse}
11:            end while
12:            for each reponse ∈ reponses do
13:                if (ComparerAvecRessourceLocale(reponse.caracteristiques) then
14:                    nbRessourcesSimilaires++
15:                end if
16:            end for each
17:            if (nbRessourcesSimilaires/|reponses|) < β then
18:                FaireConnaitreRessourceLocale()
19:            else
20:                coefficientRarete ← coefficientRarete/1.1
21:            end if
22:        end if
23:        Dormir(γ)
24:    end while
25: end procedure
```

Algorithme 12 Réception d'une requête de découverte de ressources et mise à jour de la rareté des ressources

```
1: procedure RECEPTIONREQUETEDECOUVERTERESSOURCESAVECMAJRARETE(caracteristiques,demandeur)
2:     if (ComparerAvecRessourceLocale(caracteristiques) then
3:         coefficientRarete ← coefficientRarete/1.1
4:         if RessourceLocaleEstLibre(caracteristiques) then
5:             ChangerEtatRessourceLocale(REPONSE_IMPOSSIBLE)
6:             EnvoieReponse(demandeur,caracteristiquesLocales,charge)
7:         end if
8:     end if
9: end procedure
```

du système.

Afin de rendre plus visibles ces ressources, nous proposons de les faire connaître auprès d'un certain nombre de nœuds dans le système. Pour cela, il est encore possible d'utiliser des marches aléatoires, cette fois avec une profondeur importante pour diffuser l'information. Ainsi, lorsqu'un nœud reçoit une information de ce type, il ne répond pas à l'émetteur comme lors d'une marche aléatoire classique mais enregistre cette information dans un cache. La durée de vie des informations sur les ressources rares n'a pas besoin d'être longue car nous avons vu dans l'algorithme 11 que les ressources évaluent leur rareté toutes les γ minutes. Une durée de vie légèrement supérieure à γ minutes est donc suffisante.

Par la suite, lorsqu'un nœud, initiateur ou non, reçoit une requête de découverte de ressources, il regarde dans son cache de ressources rares si un nœud peut convenir et dans ce cas il fait directement suivre cette requête de découverte de ressources vers le nœud considéré qui possède des ressources rares. Cela permet aux nœuds possédant des ressources rares d'être trouvés le pus rapidement possible.

Algorithme 13 Diffusion d'une requête permettant de faire connaître une ressource rare

```
1: procedure FAIRECONNAITRERESSOURCELOCALE
2:     noeud ← ChoisirUnVoisinAuHasard()
3:     EnvoiInformationRessourceRare(caracteristiquesRessourceLocale, TTL_depart)
4: end procedure
```

L'algorithme 13 décrit l'initiation d'une diffusion pour faire connaître une ressource rare. Nous pouvons voir que le nœud hébergeant une ressource rare n'effectue qu'une émission de requête (cf. ligne 3) sans attendre de réponse. Il suffit en effet de lancer une seule marche aléatoire avec une profondeur assez grande pour avertir TTL_{depart} nœuds dans la grille. Cette valeur n'a pas besoin d'être très grande et peut ne représenter que quelques pourcents du nombre de nœuds du système. Encore une fois, des techniques d'évaluation du nombre de nœuds dans le système peuvent être employées si une estimation n'est pas possible.

Algorithme 14 Réception, sur un nœud de la grille, d'une requête permettant de faire connaître une ressource rare

```
1: procedure RECEPTIONREQUETERESSOURCERARE(ressourceRare,TTL)
2:     if ressourceRare ∉ cacheRessourceRares then
3:         cacheRessourceRares ← cacheRessourceRares ∪ {ressourceRare}
4:     end if
5:     if TTL > 0 then
6:         noeud ← ChoisirUnVoisinAuHasard()
7:         EnvoiInformationRessourceRare(caracteristiquesRessourceLocale, TTL − 1)
8:     end if
9: end procedure
```

L'algorithme 14 présente la marche aléatoire propagée dans le système et l'algorithme 15 présente la façon dont le cache est mis à jour toutes les δ minutes.

Algorithme 15 Mise à jour du cache de ressources rares

```
1: procedure MAJCACHERESSOURCERARE
2:     while vrai do
3:         for each ressourceRare ∈ cacheRessourceRares do
4:             if (DateInsertionCache(ressourceRare) > γ) then
5:                 cacheRessourceRares ← cacheRessourceRares − {ressourceRare}
6:             end if
7:         end for each
8:         Dormir(δ)
9:     end while
10: end procedure
```

4.2.4 Réflexion sur la pertinence de l'utilisation des réseaux logiques non structurés dans un système de grille

Dans cette partie, nous avons proposé un système d'information pour grille fondé sur une architecture complètement distribuée et sur le concept de réseau pair-à-pair non structuré. Notre approche est particulièrement adaptée aux grilles composées de nœuds volatils et hétérogènes. Les réseaux pair-à-pair sont intrinsèquement adaptés à la volatilité des nœuds, à la grande échelle et ne requièrent pas de machine dédiée, fiable, puissante et onéreuse pour maintenir la cohérence du système.

Cela est toutefois à nuancer si certaines hypothèses sur la grille sont relâchées. En effet, contrairement à une approche centralisée, une approche distribuée ne permet pas d'avoir une connaissance globale des ressources qui composent la grille. Une découverte de ressources sur réseau pair-à-pair non structuré va donc statistiquement laisser de côté quelques ressources, ce qui peut engendrer une allocation de ressources moins efficace.

Les travaux récents sur les réseaux pair-à-pair structurés sont également une piste intéressante à suivre. Bénéficiant de la capacité de faire des recherches exhaustives et de bonnes propriétés de passage à l'échelle, les réseaux pair-à-pair structurés regroupent les avantages d'une architecture distribuée et d'une architecture centralisée. Il est actuellement compliqué d'ordonner dans le réseau structuré les ressources qui possèdent des attributs définis par une plage de valeur, comme la quantité de mémoire d'une ressource par exemple. Une technique pour cela consiste à utiliser une fonction de linéarisation qui permet de répartir les plages de valeurs des attributs dans une THD associée au réseau structuré. Malgré cela, il est compliqué de répartir de façon homogène les attributs des ressources dans la THD. En effet, dans les grilles la distribution du type des ressources suit une loi de Zipf [175] où beaucoup de nœuds possèdent des ressources de même type et peu sont de types différents. Ainsi, un faible nombre de nœuds aura la charge de connaître les caractéristiques de beaucoup de nœuds, ce qui pose un problème de répartition de charge. Ainsi, il en résulterait un système d'information mal équilibré et potentiellement sensible aux défaillances. Des approches prometteuses comme Mercury [56] ou Voronet [52] semblent résoudre en partie le problème moyennant un coût de maintenance, pour le moment élevé, notamment dans un environnement volatil. De plus, ces approches nécessitent que le nombre de critères soit connu, ce qui est problématique dans notre contexte puisque des nouveaux critères peuvent être ajoutés à tout moment.

4.3 Le service d'allocation de ressources

Afin d'exploiter les ressources découvertes par le système d'information, il est nécessaire de sélectionner celles qui seront finalement utilisées. C'est au service d'allocation de ressources qu'est confié l'ensemble des ressources découvertes afin qu'il puisse faire son choix, aidé pour cela d'une politique déterminée.

Nous décrivons dans un premier temps l'architecture du service d'allocation de ressources de Vigne. Ensuite nous présentons des politiques de sélection de ressources. Finalement, nous proposons des mécanismes de co-allocation de ressources.

4.3.1 Architecture du service d'allocation de ressources

Comme le montre la figure 4.6, l'architecture du service d'allocation de ressources est relativement simple.

Le service d'allocation ressources comporte quatre modules principaux qui sont le *solveur de contraintes*, le *co-allocateur de ressources*, la *sonde réseau* et *le verrouillage de ressources*.

Le solveur de contraintes permet d'appliquer une politique donnée pour choisir les meilleures ressources parmi celles découvertes.

Le co-allocateur de ressources permet d'allouer efficacement des ressources à une application composée de plusieurs tâches communicantes.

La sonde réseau dans l'architecture permet d'évaluer un lien réseau entre deux ressources. Cela est utile dans le cadre de la co-allocation de ressources pour exécuter des tâches communicantes d'une même application sur des nœuds capables de communiquer efficacement entre eux. Ce module peut aussi être utilisé par le solveur de contraintes pour certaines politiques comme par exemple celle qui viserait à exécuter des applications à proximité d'un serveur de fichier afin de minimiser le coût des transferts de données. La sonde réseau utilise directement le réseau IP et évalue la connexion en termes de bande passante et de latence.

Le verrouillage de ressources permet de supprimer temporairement du système d'information les ressources des nœuds utilisées pour l'exécution d'une application afin que ces nœuds n'allouent pas les ressources utilisées pour cette application à d'autres applications. Par exemple, si un nœud possède quatre processeurs et qu'il a été alloué à une application nécessitant trois processeurs, ce nœud ne pourra répondre favorablement qu'aux requêtes de découverte de ressources ne demandant qu'un seul processeur.

4.3.2 Politiques d'allocation de ressources

Une politique d'allocation de ressources peut être vue comme une relation d'ordre entre des ressources. Doté de cette relation d'ordre, le solveur de contraintes peut établir une liste ordonnée des ressources et donc choisir celles qui maximisent un critère défini par la politique d'allocation.

Diverses politiques peuvent être utilisées pour allouer des ressources. L'adoption d'une politique dépend du contexte d'utilisation. Nous passons en revue ici quelques politiques d'allocation de ressources en expliquant comment les mettre en œuvre dans Vigne et leur lien dans certains cas avec le système d'information.

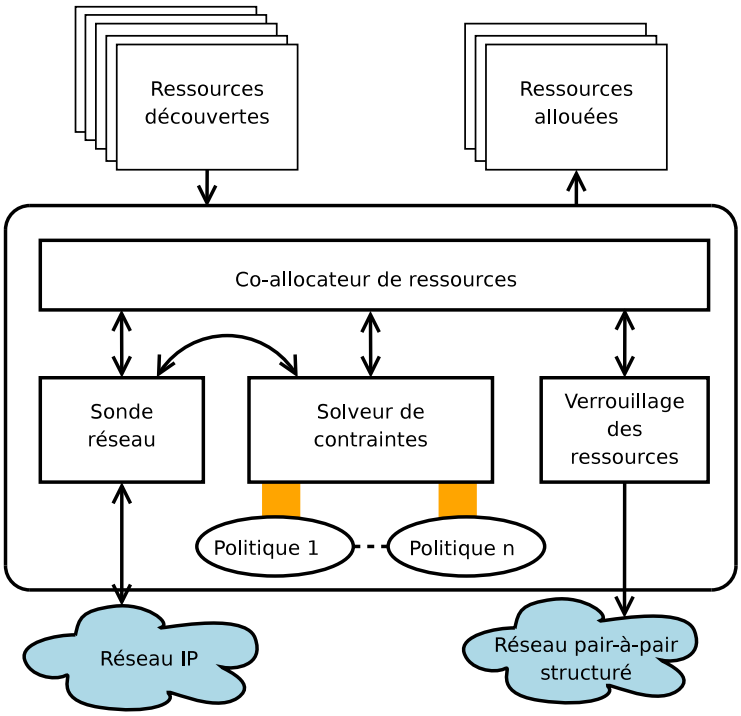


FIG. 4.6 – Architecture du service d'allocation de ressources de Vigne

4.3.2.1 Utiliser les nœuds les moins chargés

La politique d'allocation de ressources qui semble la plus naturelle est celle qui consiste à allouer les nœuds les moins chargés d'un point de vue processeur et mémoire.

Une politique visant à utiliser des nœuds peu chargés du point de vue du processeur ou de la mémoire nécessite d'avoir accès à la charge mémoire et processeur de chaque ressource découverte. Deux solutions sont possibles pour obtenir ces informations. Le service d'allocation de ressources peut contacter toutes les ressources découvertes pour leur demander leur charge ou alors le système d'information de la grille peut contenir ces informations. La première méthode a pour avantage d'obtenir des informations très à jour mais requiert des communications réseau. Avec la seconde méthode, le système d'information doit contenir des informations suffisamment *fraîches* sur la charge des ressources pour être efficace. Avec un système d'information centralisé, la seconde approche n'est pas intéressante car le coût de mise à jour du système d'information serait supérieur au coût de la première méthode. En revanche, avec un système d'information distribué, comme celui de Vigne, les informations de charge peuvent être multiplexées avec la réponse à une requête de découverte de ressources.

4.3.2.2 Modèle économique

L'utilisation des grilles de calcul dans un environnement de production soulève la question de la facturation de l'utilisation des ressources. Pour cela, les fournisseurs de ressources fixent

le prix des ressources qu'ils offrent et les consommateurs fixent le prix qu'ils sont prêts à payer.

Les politiques fondées sur un modèle économique, comme celles utilisant les critères de charge, nécessitent une extension du système d'information pour connaître le prix des ressources découvertes. Ce prix doit être régulièrement mis à jour car le prix des ressources peut varier en fonction de la loi de l'offre et de la demande.

Dans le cas de Vigne, où le système d'information est distribué, il suffit que la requête de réponse à une découverte de ressources émise par un nœud contienne le prix des ressources qu'il offre.

4.3.2.3 Minimisation des défaillances

Certaines politiques sont consacrées à la fiabilité de l'exécution des applications dans la grille et tentent par conséquent de minimiser l'impact des défaillances des ressources sur les applications. Les défaillances peuvent être d'origine matérielle ou logicielle. Les politiques visant à allouer les ressources les plus fiables possibles pourront donc évaluer la fiabilité sur deux critères.

Tout d'abord, nous pouvons admettre que passé un temps raisonnablement court, une ressource récente est moins sujette à défaillance qu'une ressource ancienne car la distribution des défaillances sur la vie d'une ressource informatique suit une loi de Weibull [131]. Afin de minimiser les conséquences des défaillances sur les applications, il est donc préférable d'allouer des ressources dont l'âge est situé, selon la distribution de Weibull, dans la période où les défaillances sont les moins probables.

Il est possible de considérer qu'une machine possédant un système d'exploitation dont la date de démarrage (*uptime*) est ancienne est une machine possédant un système stable. Même s'il est impossible de faire des hypothèses sur les ressources possédant un *uptime* faible, il peut sembler judicieux de ne pas allouer ces ressources en priorité.

L'heuristique opposée peut également être envisagée. En effet, si l'on considère qu'un système d'exploitation *vieillit* au cours du temps, il est préférable d'allouer les ressources dont l'*uptime* est le plus faible.

Pour que l'allocateur de ressources puisse prendre en compte l'âge des ressources et leur *uptime* dans sa politique de décision, il faut encore une fois que ces informations soient présentes dans le système d'information. En revanche, seul le paramètre *uptime* doit être mis à jour. Avec le système Vigne, ces informations peuvent être multiplexées dans la réponse à une requête de découverte de ressources.

4.3.2.4 Proximité d'une ressource spéciale

Certaines applications peuvent avoir un modèle de fonctionnement où d'importantes communications peuvent avoir lieu avec une ressource tierce. Cette ressource tierce peut être par exemple un serveur de fichiers stockant de gros volumes de données, un serveur de licences ou encore un instrument de mesure tel qu'un radio-télescope. Dans ce cas, il est tout à fait recommandé d'adopter une politique qui va maximiser la proximité, du point de vue du réseau, des ressources allouées pour l'application et de la ressource tierce.

Pour appliquer une politique de ce type, le service d'allocation de ressources utilise la sonde réseau permettant de déterminer les ressources possédant la meilleure connectivité avec la ressource spéciale.

4.3.2.5 Combiner des politiques d'allocations de ressources

La combinaison de plusieurs politiques est tout à fait possible, cependant il n'est pas possible de les appliquer au même niveau. C'est-à-dire que pour établir une relation d'ordre avec plusieurs politiques de découverte de ressources, il faut affecter une priorité à chaque politique.

4.3.3 Allocation de plusieurs ressources pour l'exécution d'applications communicantes

Les applications communicantes nécessitent une allocation particulière dès lors que leur exécution est sensible à la qualité du réseau. Dans ce cas, nous utilisons le module de co-allocation de ressources que nous avons introduit dans la présentation de l'architecture du service d'allocation de ressources.

La co-allocation de ressources peut être vue comme une politique prioritaire d'allocation de ressources qui vise à choisir des ressources capables de communiquer efficacement entre-elles. De plus, la co-allocation de ressources sous-entend une simultanéité de l'allocation des ressources utilisées pour des tâches communicantes. Par exemple, une application MPI nécessite que toutes les ressources utilisées par l'application soient allouées avant que l'application ne soit exécutée. En effet, démarrer une application MPI nécessite la liste de toutes les machines utilisées.

Dans cette partie, nous présentons un formalisme permettant de définir le niveau de couplage entre les tâches d'une application. Ce formalisme peut être interprété par le module de co-allocation de ressources. Nous présentons également des techniques pour établir des groupes de ressources capables de communiquer efficacement.

4.3.3.1 Définition d'un formalisme simple pour décrire le couplage entre éléments applicatifs

Nous définissons un formalisme simple permettant de décrire les groupes de tâches communicantes au sein d'une application afin de prendre en compte ce couplage dans le service d'allocation de ressources.

Soit une application $\mathscr{A}$ comportant n tâches nommées $\mathscr{T}_1, \mathscr{T}_2, ...\mathscr{T}_n$. $\mathscr{A}$ peut être définie comme l'ensemble des tâches regroupées par niveau de couplage. Nous définissons deux types de groupes qui sont le groupe des tâches non communicantes $\mathscr{G}_{nc}$ et les k groupes des tâches communicantes $\mathscr{G}_c^i$ avec $i \in [1,k]$. Nous pouvons donc noter :

$\mathscr{A} = \mathscr{G}_{nc} \cup \bigcup_{i=1}^{k} \mathscr{G}_c^i$

Au sein d'un groupe, les tâches peuvent être énumérées de cette façon :

$\mathscr{G} = \{\mathscr{T}_i, \mathscr{T}_{i+1}, ..., \mathscr{T}_{j-1}, \mathscr{T}_j\}$ avec $1 \leq i \leq j \leq n$.

Nous pouvons aussi noter $\mathscr{G} = \{\mathscr{T}_i \rightarrow \mathscr{T}_j\}$ pour désigner toutes les tâches de $\mathscr{T}_i$ à $\mathscr{T}_j$.

Voyons comment les différents types d'applications introduits dans la partie 2.1.3 sont représentés avec ce formalisme.

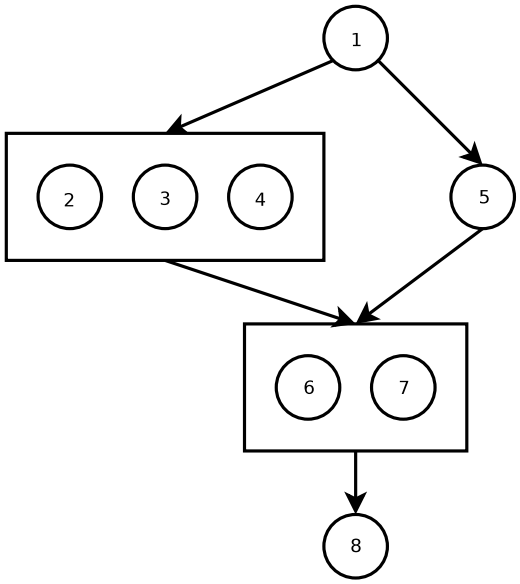


FIG. 4.7 – Exemple d'application composée de plusieurs modèles d'application

Sac de tâches indépendantes et applications maître/travailleurs Avec ce type d'application, il n'existe aucun couplage entre les tâches. Ainsi,
$\mathscr{A} = \mathscr{G}_{nc}$ avec $\mathscr{G}_{nc} = \{\mathscr{T}_1 \rightarrow \mathscr{T}_n\}$.

Applications parallèles Avec ce type d'application, toutes les tâches sont couplées. Ainsi,
$\mathscr{A} = \mathscr{G}_c^1$ avec $\mathscr{G}_c^1 = \{\mathscr{T}_1 \rightarrow \mathscr{T}_n\}$.

Composition de modèles d'application La figure 4.7 représente une application dont l'architecture est composée de plusieurs modèles existants. Cette application est composée de deux groupes de tâches parallèles et de trois tâches non communicantes. Cette application peut se décrire ainsi :
$\mathscr{A} = \mathscr{G}_{nc} \cup \mathscr{G}_c^1 \cup \mathscr{G}_c^2$ avec $\mathscr{G}_{nc} = \{\mathscr{T}_1, \mathscr{T}_5, \mathscr{T}_8\}, \mathscr{G}_c^1 = \{\mathscr{T}_2 \rightarrow \mathscr{T}_4\}, \mathscr{G}_c^2 = \{\mathscr{T}_6, \mathscr{T}_7\}$.

4.3.3.2 Détermination de la proximité entre des ressources

Afin d'appliquer une co-allocation d'application décrite par le formalisme présenté précédemment, le service d'allocation de ressources utilise la sonde réseau afin de déterminer les caractéristiques du réseau entre les ressources qui ont été découvertes. La qualité du réseau entre les nœuds est utilisée pour choisir au mieux les ressources qui hébergent des tâches fortement communicantes.

Nous proposons deux méthodes pour que la sonde réseau détermine la distance entre des ressources.

La première méthode consiste à construire la matrice des latences réseau entre les ressources découvertes. Pour cela, une communication doit être établie entre chaque paire de

ressources afin d'évaluer la latence réseau pour chacune de ces paires. Si la latence peut être considérée comme identique dans les deux sens des canaux de communication, il suffit de faire $\sum_{i=1}^{n-1} i$ mesures avec n le nombre de ressources (ce qui correspond au nombre d'éléments au-dessus ou en dessous de la diagonale d'une matrice). Si la latence ne peut pas être considérée comme identique dans les deux sens des canaux de communication, alors il faut faire $2 \times \sum_{i=1}^{n-1} i$ mesures. Dans ce dernier cas, le co-allocateur de ressources ne retiendra que la valeur la plus élevée entre chaque paire de ressources. Dans certains schémas applicatifs, il serait également intéressant de prendre en compte le débit possible entre deux nœuds pour des tâches qui échangent de grandes quantités de données. Toutefois, cette métrique est complexe à évaluer car l'occupation d'un lien réseau peut évoluer dans le temps. Il est donc nécessaire de disposer d'une infrastructure permettant la réservation de liens réseaux avec des mécanismes de qualité de service comme RSVP/IntServ [171].

Une seconde heuristique consiste à utiliser les noms DNS (*Domain Name System*) des ressources. Dans le cas simple, nous supposons que plus le suffixe DNS partagé entre deux ressources est grand, plus il est probable que ces ressources soient proches. Par exemple, `parasol22.rennes.grid5000.fr` sera probablement plus proche de `paravent51-.rennes.grid5000.fr` que de `grillon16.nancy.grid5000.fr` puisqu'elles ont le suffixe `.rennes.grid5000.fr` en commun alors que dans l'autre cas, seul le suffixe `.grid5000.fr` est commun. Dans certains cas, le schéma de désignation des ressources ne suit pas la convention DNS. Par exemple, `paravent51.rennes.grid5000.fr` pourrait s'appeler `rennes-paravent51.grid5000.fr`. Dans ce cas, il est impossible de déterminer la proximité de ressources en utilisant le plus grand suffixe commun. Une solution est de demander au fournisseur des ressources de donner le schéma de désignation des ressources qu'il propose et de l'inclure dans la description locale des ressources. Dans l'exemple, le fournisseur de ressources pourrait fournir le motif `%3-%4.%2.%1` où `%1` décrit le domaine le plus large et `%n` décrit la dernière sub-division. Muni de ce motif, le co-allocateur de ressources peut établir un ordre parmi les ressources qui possèdent le même motif. Si les motifs entre deux ressources sont différents, il suffit de ne prendre en compte que le plus grand suffixe commun dans le nom DNS car cela veut dire que les ressources ne sont pas gérées par le même domaine d'administration.

Nous avons vu dans la figure 3.6 que les communications dans le système Vigne reposent sur des réseaux logiques pair-à-pair. Cette approche occulte le nom réel d'une ressources puisqu'un nom logique est utilisé. Par conséquent, pour appliquer la seconde méthode, le système d'information doit comporter le nom DNS des ressources. De la même façon que pour les informations de charge, le nom DNS peut être multiplexé dans une réponse à une requête de découverte de ressources.

4.4 Conclusion

Dans ce chapitre, nous avons présenté le système d'information et le service d'allocation de ressources du système Vigne. Nous avons tout d'abord survolé les objectifs de ces deux services en montrant qu'ils sont liés. En effet, la pertinence des informations obtenues par le système d'information garantit une application efficace des politiques d'allocation de ressources.

Ensuite, nous avons détaillé le fonctionnement du système d'information de Vigne. Ce dernier est fondé sur l'utilisation d'un réseau pair-à-pair non structuré et du concept de marches aléatoires. Nous avons présenté une extension du concept de marches aléatoires permettant d'augmenter la pertinence des informations obtenues à l'issue d'une découverte de ressources pour une utilisation dans le cadre de l'allocation des ressources d'une grille. Cette proposition repose sur l'utilisation de caches gérés avec une politique favorisant les nœuds possédant des ressources non allouées et sur un mécanisme de dissémination d'informations permettant d'améliorer de façon significative le remplissage des caches. De plus, nous proposons un mécanisme permettant de combler une lacune inhérente à l'utilisation d'un réseau pair-à-pair non structuré qui est la découverte de ressources rares. Dans l'état de l'art, les projets Vishwa [168] et Zorilla [73] fondent leur système d'information sur un réseau non-structuré et sur le concept d'inondation basique. Toutefois, ils ne proposent aucune optimisation pour limiter le coût de l'inondation et pour améliorer la découverte de nœuds possédant des ressources libres. Ils ne proposent pas non plus de solution pour découvrir plus facilement les ressources rares.

Nous avons détaillé le fonctionnement du service d'allocation de ressources de Vigne. Nous avons étudié les interactions nécessaires avec le système d'information et l'architecture du service d'allocation pour mettre en œuvre différentes politiques d'allocation de ressources. Puis, nous avons proposé un formalisme permettant de décrire le couplage entre les tâches d'une application afin que le module de co-allocation puisse allouer des ressources ayant une connectivité réseau efficace pour une application composée de tâches qui sont fortement communicantes. Nous avons aussi proposé deux méthodes pour permettre au module de co-allocation d'évaluer la connectivité réseau entre les nœuds de la grille.

Nous avons présenté deux services possédant individuellement d'excellentes propriétés pour une utilisation dans le cadre d'une grille de grande taille. Tout d'abord, nous avons proposé un système d'information distribué, tolérant aux défaillances multiples et adapté à la grande échelle d'une grille. Ensuite, nous avons proposé un service d'allocation de ressources permettant de mettre en œuvre les politiques d'allocation de ressources les plus communes et permettant de co-allouer des ressources de façon efficace en minimisant la distance réseau des ressources allouées à des tâches communicantes. Grâce à la coopération entre ces deux services, le système Vigne possède des fondations solides pour l'exécution fiable et efficace d'applications sur une grille de calcul. L'évaluation du système d'information et du service d'allocations de ressources est présentée dans le chapitre 6.

Chapitre 5

Exécution simple, efficace et fiable d'applications sur la grille

Nous présentons dans ce chapitre les contributions que nous avons apportées à l'exécution d'applications sur une grille de grande taille. Dans un premier temps, nous décrivons dans sa globalité le service de gestion d'application de Vigne. Ensuite nous présentons les mécanismes de Vigne qui permettent de superviser l'exécution des applications afin de garantir une exécution fiable.

5.1 La gestion des applications

Nous proposons un service de gestion d'application permettant de concilier simplicité et efficacité pour l'exécution des applications sur une grille. Pour simplifier l'exécution d'applications, le service est doté d'un gestionnaire de cycle de vie qui prend en charge l'ensemble des opérations (décrites dans la partie 3.3) nécessaires au déroulement de l'exécution une fois que l'application est soumise. Pour garantir une exécution efficace, le service de gestion d'application est fondé sur les deux services présentés dans le chapitre 4 qui permettent de répartir la charge applicative sur l'ensemble des ressources de la grille tout en tenant compte des contraintes imposées par les applications composées de tâches communicantes.

Dans cette partie, nous décrivons tout d'abord l'architecture du service de gestion d'application de Vigne. Ensuite, nous détaillons la gestion du cycle de vie d'une application exécutée dans une grille Vigne. Finalement, nous présentons les fonctionnalités proposées par Vigne pour exécuter des applications distribuées sur une grille.

5.1.1 Architecture du service de gestion d'application

Comme le système d'information et le service d'allocation de ressources, le service de gestion d'application de Vigne est complètement distribué.

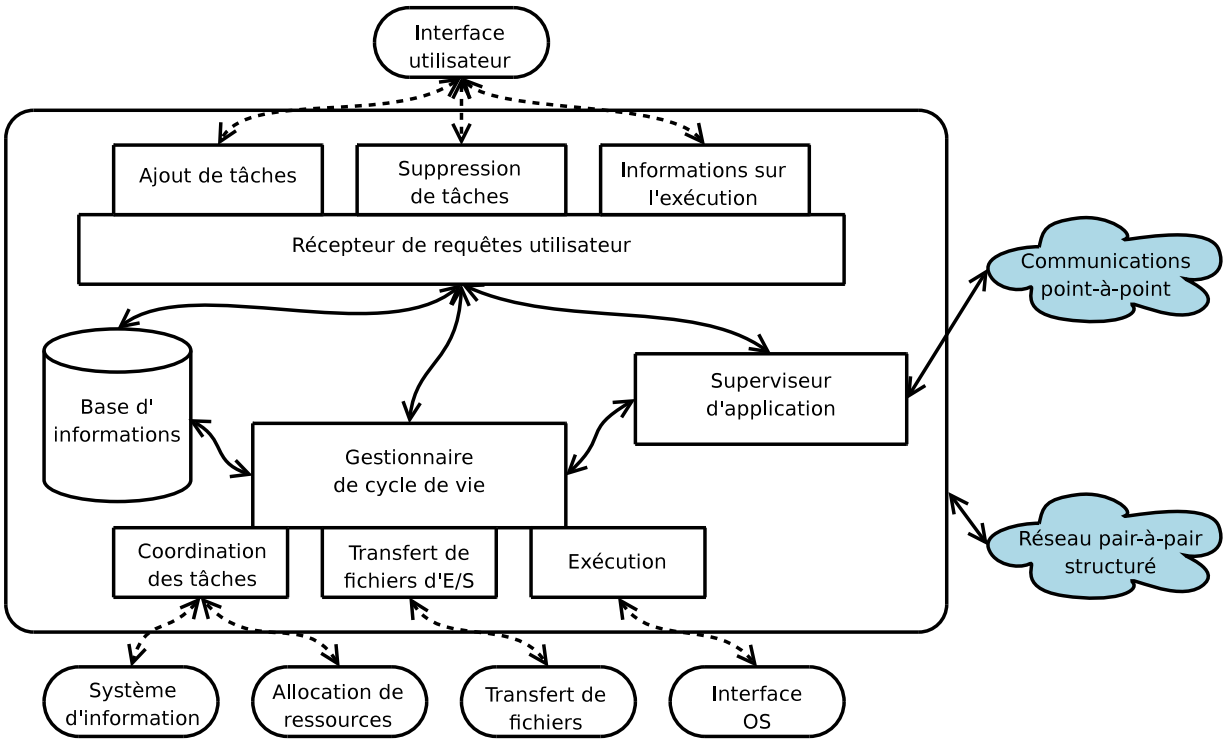


FIG. 5.1 – Architecture du service de gestion d'application

5.1.1.1 Vue globale de l'architecture

La figure 5.1 présente l'architecture logicielle du service de gestion d'application qui est présent sur chaque nœud de la grille. Le service de gestion d'application peut être vu comme l'interface entre les utilisateurs et les applications. Il possède trois parties principales qui sont : la base d'informations sur les applications, le récepteur des requêtes des utilisateurs et le gestionnaire du cycle de vie des applications.

La base d'informations sur les applications contient toutes les informations relatives aux tâches des applications. Toutes les applications gérées par le nœud considéré y sont donc enregistrées et pour chaque application, l'ensemble de ses tâches y est également enregistré. Les informations enregistrées pour une tâche sont par exemple : le type des ressources nécessaires à son exécution, les ressources réellement utilisées pour son exécution, l'état d'avancement dans l'exécution, les fichiers d'entrée qu'elle utilise, les fichiers de sortie qu'elle produit en local ou encore les dépendances qu'elle possède avec les autres tâches de l'application.

Les requêtes des utilisateurs concernent des ajouts de tâches, des suppression de tâches ou des demandes d'information sur l'exécution d'une application. Ces requêtes peuvent modifier l'état des tâches des applications ou encore provoquer des actions sur le gestionnaire de cycle de vie.

Le gestionnaire de cycle de vie est chargé de mener à bien l'exécution d'une application en coordonnant les différentes tâches d'une application, en transférant les fichiers d'entrée/sortie

des tâches ou encore en exécutant de façon fiable les tâches. Pour ce dernier point, le gestionnaire de cycle de vie utilise un superviseur d'applications qui permet de détecter les défaillances des tâches et des nœuds utilisés par les tâches exécutées.

5.1.1.2 Vue distribuée du service de gestion d'application

Lorsqu'une application est soumise au système Vigne, un identifiant lui est attribué. Cet identifiant est obtenu en calculant le hachage SHA-1 [74] du triplet *{heure de soumission, nom de la machine utilisée pour la soumission, hachage MD5 [146] du premier binaire de l'application}*. Compte-tenu de la probabilité de collision très faible entre deux éléments hachés avec un algorithme comme SHA-1, l'identifiant d'une application peut être considéré comme unique. Dans le cas où l'identifiant aurait malgré tout été attribué et qu'une application serait déjà enregistrée dans la base d'informations avec cet identifiant, le système diffère la soumission d'une seconde, et cela autant de fois que nécessaire, ce qui générera un nouvel identifiant pour l'application. Cet identifiant unique est utilisé pour classer les informations liées à une application dans la base d'informations.

L'information relative aux applications dans la grille est distribuée sur plusieurs nœuds de la grille afin d'éviter les problèmes liés à l'utilisation d'un serveur central. Pour cela, les informations liées à une application sont hébergées par le nœud qui possède l'identifiant dans le réseau structuré le plus proche numériquement de l'identifiant de l'application. Le nœud contenant les informations d'une application peut être contacté simplement par envoi de messages par routage dans le réseau structuré. Cela permet donc de retrouver les informations relatives à une application sans connaître leur localisation physique. Nous voyons dans la partie 5.1.1.3 que cela est une première étape pour garantir une tolérance aux défaillances complète du service de gestion d'application.

Nous définissons l'expression *gestionnaire d'application* qui détermine une instance du service de gestion d'application sur un nœud accompagnée des informations concernant une application. Ainsi, le service de gestion d'application est composé de l'ensemble des gestionnaires d'application répartis sur la grille, où chaque gestionnaire d'application est chargé de l'exécution d'une application. Un gestionnaire d'application est représenté par une clé de la THD utilisée pour la gestion d'application et mise en œuvre sur le réseau structuré de Vigne. Cette clé a pour nom l'identifiant de l'application. Si le nombre de nœuds dans le système est inférieur au nombre maximal de clés de la table de hachage distribuée, ce qui semble raisonnable puisque le nom des clés peut être de l'ordre de 2^{128} ou 2^{256}, un nœud est chargé de gérer plusieurs clés et donc potentiellement plusieurs gestionnaires d'application.

La figure 5.2 présente un exemple de grille où deux applications ont été lancées. Il y a donc deux gestionnaires d'application (représentés par des carrés) : le gestionnaire `E4` est localisé sur le nœud `D7` et gère deux tâches (représentées par des étoiles) sur les ressources `3A` et `53`, le gestionnaire `81` est localisé sur le nœud `A0` et gère une tâche sur la ressource `0F`.

Grâce à cette architecture distribuée, le service de gestion d'application est réparti sur les nœuds de la grille. Les propriétés d'un algorithme de hachage tel que SHA-1 assurent une distribution homogène des éléments de l'espace de départ dans l'espace d'arrivée. Ainsi, les identifiants d'applications sont répartis de façon homogène sur l'espace des clés, ce qui assure que les gestionnaires d'application sont répartis de façon uniforme sur les nœuds de la grille, si

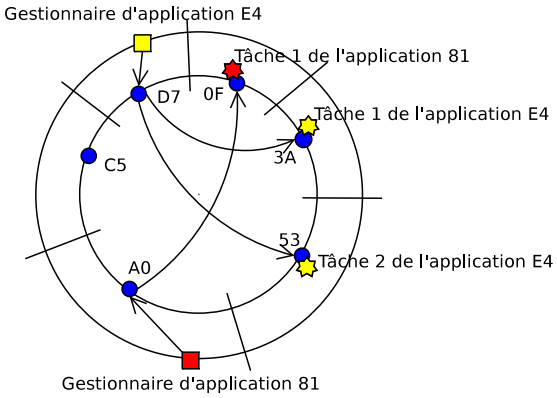


FIG. 5.2 – Exemple de grille avec deux gestionnaires d'application

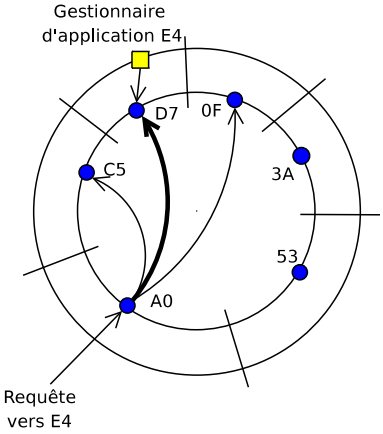


FIG. 5.3 – Exemple de routage de message avec duplication ($n = 1$) dans le réseau structuré

tant est que les noms des nœuds soient répartis de façon uniforme dans l'espace de désignation. Pour cela, lorsqu'un nœud est introduit dans la grille, le système lui attribue un nom adéquat en cherchant le plus grand écart numérique existant entre les nœuds et en choisissant un nom à la moyenne arithmétique des nœuds bordant cet écart.

5.1.1.3 Tolérance à la volatilité et aux défaillances multiples du service de gestion d'application

Le service de gestion d'application et en particulier la base d'informations sont des points critiques du système Vigne. Si les informations relatives à une application sont perdues, l'application est perdue. Ainsi, lorsqu'un nœud se retire de la grille ou s'il subit une défaillance, les informations relatives aux applications doivent survivre.

Louis Rilling a proposé dans [143, 144] un mécanisme général pour résoudre le problème de la défaillance de nœuds en utilisant une forme de duplication active des informations. Voyons comment il s'applique à la survie du service de gestion d'application de Vigne.

Cette forme de duplication active repose sur le concept de diffusion (*multicast* en anglais) atomique dans le réseau structuré. Les messages routés dans le réseau structuré peuvent être automatiquement envoyés aux voisins du nœud initialement destinataire. Ainsi toutes les informations relatives à une application sont présentes sur $2n+1$ nœuds où n est le nombre de voisins de chaque côté du nœud principal sur lesquels sont dupliqués les informations. Dans ce cas, $2n$ défaillances simultanées peuvent être tolérées. La figure 5.3 montre un exemple de routage avec duplication des messages sur 3 nœuds ($n = 1$). Dans cet exemple, un message est routé du nœud `A0` vers le nœud responsable de l'application `E4`, c'est-à-dire le nœud `D7`. Le message est également routé vers un voisin de chaque côté, dans l'anneau logique, du nœud `D7`, c'est-à-dire vers les nœuds `C5` et `0F`.

Sachant que les messages sont envoyés à plusieurs nœuds, plusieurs nœuds peuvent répondre aux requêtes. Tous les services du système Vigne supportent la réception de messages

dupliqués puisqu'ils sont fondés sur un automate d'état déterministe. Dans le cas où un nœud reçoit un message qui a déjà été reçu, il ignore le message. Un exemple d'automate est présenté dans la partie 5.1.3.3.

Voyons maintenant comment peuvent être traitées les défaillances ou la volatilité des nœuds. Reprenons l'exemple de la figure 5.3. Si un nœud du groupe multicast cible se retire de la grille ou s'il subit une défaillance, le nombre de $2n$ défaillances simultanées supportées doit être garanti. Il faut donc qu'un nouveau nœud soit inclus dans le groupe multicast. Cela ne pose pas vraiment de problème car le routage peut garantir que $2n+1$ nœuds recevront les messages. Le problème avec ce nouveau nœud est qu'il ne possède pas un état identique à celui des autres nœuds du groupe.

Algorithme 16 Réception par multicast d'un message de gestion d'application

```
1: procedure RECEPTIONMESSAGEGESTIONAPPLICATION(idApplication, msg)
2:     if idApplication ∈ baseLocaleInformations then
3:         Traitement(idApplication, msg)
4:     else
5:         EnvoiMulticastDemandeEtat(idApplication)
6:     end if
7: end procedure
```

L'algorithme 16 présente le traitement qui est réalisé par un nœud qui reçoit un message concernant le service de gestion d'application. Si l'identifiant d'application est connu du nœud, alors ce dernier traite le message. Dans le cas contraire il envoie une requête au groupe multicast relatif à l'application pour obtenir un état cohérent.

Algorithme 17 Réception par multicast d'une demande d'état

```
1: procedure RECEPTIONDEMANDEETAT(idApplication)
2:     if idApplication ∈ baseLocaleInformations then
3:         EnvoiMulticastEtat(etatLocal)
4:     end if
5: end procedure
```

Lorsque les nœuds d'un groupe multicast reçoivent une telle demande, ils effectuent le traitement décrit par l'algorithme 17. Ils vérifient qu'ils possèdent des informations sur l'application concernée dans leur base locale. Les nœuds possédant un état l'envoient au groupe multicast. Les nœuds ne possédant pas d'état, qui sont par conséquent les demandeurs, n'ont rien à faire. Sur réception de plusieurs messages contenant un état (puisque plusieurs nœuds du groupe multicast peuvent avoir envoyé leur état), les nœuds demandeurs ne tiennent compte que du premier message. Cela est suffisant puisque par construction du groupe multicast, la cohérence atomique est garantie entre tous les états, qui sont par conséquent identiques.

Les figures 5.4, 5.5 et 5.6 présentent un exemple de demande d'état. Considérons un groupe multicast composé des nœuds 1, 2 et 3. Les nœuds 1 et 2 (représentés en trait plein) connaissent l'état d'une application et le nœud 3 (représenté en pointillés) ne le connait pas encore. La figure 5.4 montre la façon dont un nœud quelconque, noté R, envoie une requête au service de gestion d'application. Nous pouvons voir que les nœuds 1, 2 et 3 reçoivent la requête. Dans la figure 5.5, nous pouvons voir que les nœuds 1 et 2, qui possèdent l'état de l'application,

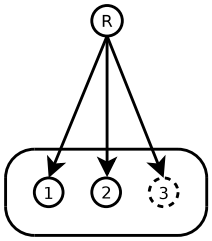


FIG. 5.4 – Demande d'état, étape 1

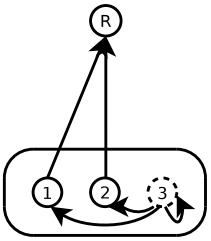


FIG. 5.5 – Demande d'état, étape 2

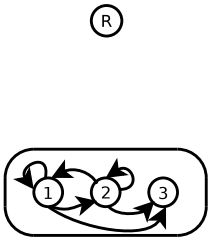


FIG. 5.6 – Demande d'état, étape 3

répondent à R. Quant au nœud 3, il ne répond pas. En revanche, il envoie au groupe multicast une demande d'état. Dans la figure 5.6, nous pouvons voir que les nœuds possédant l'état de l'application envoient cet état à tout le groupe multicast. C'est ainsi que le nœud 3 (maintenant représenté en trait plein) possède l'état de l'application et qu'il est apte à répondre aux requêtes futures.

Ce mécanisme permet de traiter indifféremment le départ du nœud responsable de la clé de l'application et le départ des nœuds voisins.

Si un nœud s'insère dans l'espace numérique couvert par les nœuds d'un groupe multicast, alors il doit récupérer l'état des gestionnaires d'application dupliqués en suivant l'algorithme 16 et un nœud du groupe doit être évincé. En effet, suite à l'insertion d'un nœud dans l'espace numérique couvert par les nœuds d'un groupe multicast, un nœud ayant un identifiant se situant en bordure de l'espace numérique ne recevra plus les requêtes adressées au groupe. Voici comment nous gérons l'éviction d'un tel nœud. Nous supposons que pour qu'un nœud participe à un groupe multicast, c'est-à-dire pour qu'il réponde aux requêtes destinées au groupe, il doit recevoir des messages régulièrement. Ainsi, si un nœud ne reçoit plus de messages de multicast depuis un certain temps, il considère qu'il n'appartient plus au groupe et il supprime toutes les informations relatives au gestionnaire d'application qui était dupliqué.

5.1.2 La gestion du cycle de vie d'une application mono-tâche

Lorsqu'une application est soumise à une grille gérée par le système Vigne, le service de gestion d'application prend en charge le cycle de vie complet de l'application. Dans cette partie, nous abordons seulement le cas simple où une application est composée d'une seule tâche. Nous étudions dans la partie suivante comment sont traitées les applications composées de multiples tâches.

Le gestionnaire du cycle de vie (GCV) d'une tâche peut être vu comme une unité de pilotage de tous les services du système.

La figure 5.7 présente l'automate d'état du GCV d'une application mono-tâche. Nous pouvons voir que lorsqu'une demande d'exécution est faite au GCV, ce dernier interroge le système d'information pour obtenir des ressources libres pouvant convenir à l'exécution de la tâche. Si le gestionnaire d'application est dupliqué, plusieurs requêtes d'interrogations sont envoyées au système d'information. Toutefois, ce dernier ne traite que la première qu'il reçoit pour une application considérée afin d'éviter la multiplication des requêtes de découverte de ressources

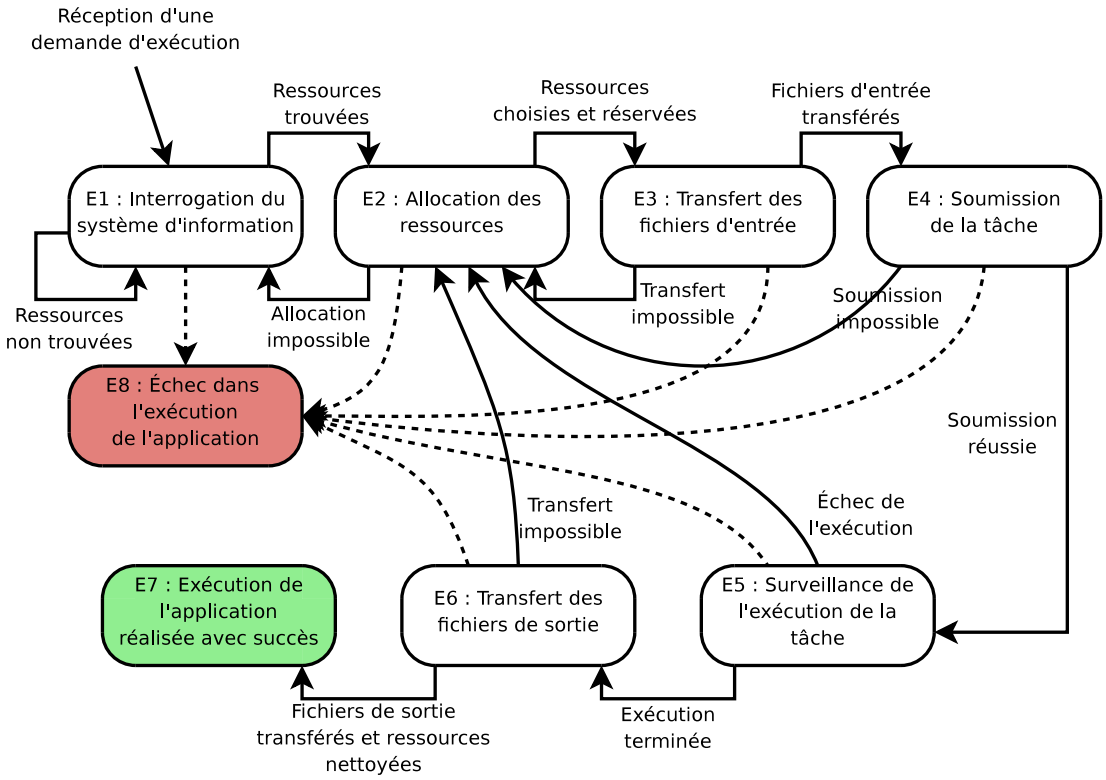


FIG. 5.7 – Automate d'état du gestionnaire de cycle de vie pour une application mono-tâche

dans le réseau non structuré. Le nœud responsable de la découverte de ressources est celui qui possède l'identifiant numérique le plus proche de celui de l'application. Si ce nœud subit une défaillance, le GCV interroge une nouvelle fois le système d'information, ce qui aboutit à la sélection d'un nouveau nœud responsable de la découverte de ressources. Lorsque des ressources sont trouvées, le GCV les transmet au service d'allocation des ressources. Si des ressources peuvent être allouées, elles sont réservées et le GCV initie le transfert des fichiers d'entrée de l'application. Une fois les fichiers d'entrée transférés, l'application est soumise et le GCV est chargé de surveiller son exécution. Lorsque l'exécution de l'application est terminée, le GCV transfère les fichiers de sortie de l'application et ordonne la libération des ressources qui ont été utilisées. Les éventuels fichiers temporaires ou processus subsistant à une exécution sur un nœud comme par exemple un démon MPI ou une machine virtuelle Java sont supprimés. Une fois toutes ces étapes franchies avec succès, le GCV est dans l'état E7, ce qui correspond à la fin du cycle de vie de l'application.

Voyons maintenant comment sont gérés les cas d'erreurs qui peuvent se produire. Dans l'état E1, il est possible que l'interrogation du système d'information échoue si un nombre insuffisant de ressources est trouvé. Dans ce cas, le GCV reste en E1 pour effectuer de nouvelles tentatives d'interrogation. Dans l'état E2, il est possible que l'allocation des ressources soit impossible et que le GCV retourne dans l'état E1 pour que le système trouve d'autres ressources. Dans l'état E3, il est possible que les fichiers d'entrée n'aient pas pu être transférés correcte-

ment pour diverses raisons (espace disque plein, défaillance réseau, etc). Dans ce cas le GCV repasse dans l'état E2 pour que le système alloue de nouvelles ressources. Dans l'état E4, si la soumission de la tâche échoue, le GCV retourne également dans l'état E2. Il en est de même pour l'état E5 si une défaillance se produit pendant l'exécution de l'application et pour l'état E6 si le transfert des fichiers de sortie ne peut pas s'effectuer convenablement. Il est ainsi possible que le GCV repasse dans un état antérieur en cas d'échec. Toutefois, un certain nombre d'échecs sont tolérés mais le système peut décider que l'échec est dû au code de l'application. Dans ce dernier cas, le GCV passe dans l'état E8 qui correspond à l'échec de l'exécution de l'application (flèches en pointillés sur la figure 5.7). D'une façon générale, l'état E8 n'est atteint que si aucune ressource ne peut être trouvée pour exécuter l'application, si une défaillance d'un serveur de fichiers se produit ou si l'application comporte une erreur de programmation qui conduit à un échec de l'exécution.

5.1.3 La gestion du cycle de vie d'une application multi-tâche

La gestion du cycle de vie des applications multi-tâche est moins triviale que pour les applications mono-tâche. Dans le système Vigne, chaque tâche d'une application est gérée par un GCV. Ainsi une application multi-tâche comporte plusieurs GCV qui, nous allons le voir, doivent parfois interagir. Nous examinons dans cette partie les particularités de chaque modèle d'applications, les fonctionnalités requises pour l'exécution de ces applications et l'automate complet du gestionnaire de cycle de vie des applications.

5.1.3.1 Caractéristiques des modèles d'applications multi-tâche

Nous rappelons ici les caractéristiques des applications que nous souhaitons prendre en charge dans notre système de grille, c'est-à-dire celles présentées dans la partie 2.1.3.

Les applications de type *sac de tâches* sont les plus simples et ne requièrent pas de fonctionnalités particulières du système si ce n'est une allocation de multiples ressources. Dans ce cas, chaque tâche est gérée par un gestionnaire de cycle de vie indépendant, comme celui présenté dans la partie 5.1.2, qui s'assure de sa terminaison. Ainsi, cela garantit la terminaison globale de l'application.

Les applications de type *maître/travailleurs* nécessitent une allocation simultanée des ressources utilisées par le maître et les travailleurs pour que le maître puisse distribuer les tâches aux travailleurs. Nous pouvons définir ce pré-requis comme une *dépendance de synchronisation* entre les tâches pour l'allocation des ressources.

Les applications *parallèles* nécessitent également une allocation simultanée des ressources. Par exemple, dans le paradigme MPI, un fichier `machinefile` contenant tous les nœuds doit être établi pour lancer l'application avec la commande `mpiexec`. D'autres part, les applications parallèles sont souvent composées de tâches qui communiquent intensivement. Ainsi les performances de l'application sont supérieures si les tâches sont exécutées sur des ressources proches en termes de connectivité réseau. Nous pouvons définir cela comme une *dépendance spatiale* entre les tâches. Les applications *distribuées* peuvent aussi être composées de tâches ayant des dépendances de synchronisation et spatiales bien que cela ne soit pas systématique.

Les applications de type *couplage de code* sont composées de tâches communicantes ou

de groupes de tâches communicant entre eux. Ainsi, les tâches qui composent ce type d'application possèdent des dépendances de synchronisation et éventuellement des dépendances spatiales s'il y a des groupes de tâches parallèles ou si les tâches couplées sont fortement communicantes.

Finalement, les applications de type *workflow* sont un assemblage des autres types d'applications. Des groupes de tâches d'un *workflow* peuvent avoir des dépendances de synchronisation et spatiales. De plus les groupes de tâches sont liés par une dépendance que nous pouvons définir comme une *dépendance de précédence*. En effet, les résultats de certaines tâches sont requis pour que d'autres tâches puissent démarrer.

Type d'application \ Type de dépendance	Synchronisation	Spatiale	Précédence
Sac de tâches	✘	✘	✘
Maître/esclave	✔	✘	✘
Distribué/Parallèle	✔	✔	✘
Couplage de codes	✔	✔	✘
Workflow	✔	✔	✔

TAB. 5.1 – Dépendances entre les tâches de différents types d'applications

Le tableau 5.1 résume les dépendances qui peuvent exister entre les tâches d'une application en fonction de son modèle. Nous souhaitons que le gestionnaire de cycle de vie des applications offre des fonctionnalités permettant de prendre en charge les trois types de dépendances présentés afin de supporter ces modèles d'applications.

5.1.3.2 Fonctionnalités système nécessaires à l'exécution d'une application multi-tâche

Allocation synchronisée de plusieurs ressources Afin de satisfaire les dépendances de synchronisation entre tâches, le service de gestion d'application doit être capable d'allouer de façon synchronisée plusieurs ressources pour exécuter en même temps les différentes tâches concernées.

Pour exécuter de façon synchronisée différentes tâches qui possèdent une dépendance de synchronisation, le GCV de chaque tâche initie de façon indépendante la découverte et l'allocation des ressources, puis entre dans une barrière de synchronisation. Il est nécessaire d'initier les découvertes de ressources de façon indépendante car chaque tâche peut nécessiter des ressources différentes. Lorsque tous les GCV d'un groupe de tâches possédant une dépendance de synchronisation sont dans la barrière, le GCV de chaque tâche poursuit le pilotage de la tâche dont il est responsable.

L'algorithme 18 décrit les opérations réalisées lorsque le GCV d'une tâche appartenant à un groupe de tâches liées par une dépendance de synchronisation arrive à la fin de l'état où les fichiers d'entrée sont transférés. Le GCV vérifie si toutes les tâches du groupe ont terminé l'étape d'allocation de ressources (lignes 5 à 10) et si tel est le cas, cela veut dire qu'il est le dernier GCV du groupe à entrer dans la barrière. Il pourra ainsi notifier tous les GCV du groupe qui étaient en attente (lignes 12 à 14) afin qu'ils poursuivent le pilotage des tâches dont ils sont

responsables.

Algorithme 18 Barrière de synchronisation d'un groupe de tâches qui possèdent une dépendance de synchronisation

```
 1: procedure BARRIERESYNCHROGROUPETACHES(tache, groupe)
 2:     liste ← ObtenirListeDesTachesDuGroupe(groupe)
 3:     tachesPretes ← vrai
 4:     i ← 0
 5:     while (i < taille(liste)) ∧ (tachesPretes = faux) do
 6:         t ← liste[i]
 7:         if (t ≠ tache) ∧ (t.etat ≠ ALLOCATION_TERMINEE) then
 8:             tachesPretes ← faux
 9:         end if
10:     end while
11:     if tachesPretes = vrai then
12:         for each t ∈ liste do
13:             DemanderPassageEtatSoumissionTache(t)
14:         end for each
15:     else
16:         /* nous ne faisons rien ici, il suffit d'attendre */
17:     end if
18: end procedure
```

Allocation de ressources proches du point de vue du réseau Afin de satisfaire les dépendances spatiales entre les tâches d'une application, spécialement pour les groupes de tâches communicantes, nous utilisons les mécanismes de co-allocation présentés dans la partie 4.3.3.

Sachant que les découvertes de ressources sont effectuées de façon indépendantes pour chaque tâche, deux mécanismes permettent de maximiser les chances de trouver des ressources proches pour exécuter un groupe de tâches communicantes.

Tout d'abord, lorsqu'une requête liée à la découverte de ressources pour l'exécution d'une tâche communicante, le nombre de résultats demandés à l'issue de la découverte de ressources est plus grand que pour un autre type de requête. En effet, il est plus probable de trouver des nœuds proches parmi un choix plus grand.

Ensuite, le protocole de découverte de ressources est optimisé pour le cas où une découverte de ressources est initiée pour une tâche communicante. Dans la partie 4.2.2.4 (algorithme 8), nous avons vu que lorsqu'un nœud répondait à une requête, il ne pouvait pas répondre à une autre pendant un certain temps. Toutefois, le protocole se comporte différemment lorsque plusieurs requêtes concernant les tâches communicantes d'une même application sont initiées. Dans ce cas, le nœud qui a récemment répondu à une requête peut tout de même répondre à une autre requête qui concerne une tâche communicante de la même application. Puisqu'un seul résultat par découverte de ressources est utilisé pour une tâche, cette optimisation est pertinente et permet d'augmenter le nombre de résultats retournés pour une découverte de ressources. Sachant que les résultats des différentes requêtes sont traités par les GCV du même gestionnaire d'application, un mécanisme permet d'empêcher l'allocation d'une même ressource à plusieurs tâches.

Environnement d'exécution Un certain nombre d'applications multi-tâche dont les applications parallèles ont besoin de connaître avant leur exécution les ressources qui seront utilisées. Par exemple, dans le cas d'une application parallèle suivant le paradigme MPI, la liste des ressources doit être fournie à l'exécution.

Compte-tenu de la vue SIU que fournit Vigne pour masquer à l'utilisateur la distribution des ressources utilisées pour une exécution et comme nous supposons que les applications ne sont pas modifiées pour être exécutées dans Vigne, une alternative doit être utilisée pour que les applications composées de groupes de tâches communicantes puissent être exécutées.

Pour cela, le système Vigne modifie l'environnement d'exécution de chaque tâche pour que les nœuds utilisés par les autres tâches puissent être connus.

Nous définissons ainsi trois types de variables d'environnement. Le premier type, de la forme `VIGNE_TACHE_n` correspond au nœud utilisé par la $n^{ième}$ tâche de l'application. Le second type de variable d'environnement, de la forme `VIGNE_GROUPE`, contient le chemin d'un fichier rempli automatiquement avec l'ensemble des nœuds utilisés par un groupe de tâches. Cela correspond typiquement au fichier `machinefile` utilisé par une application MPI. Le dernier type de variable d'environnement, de la forme `VIGNE_GROUPE_TACHE_n` contient le nœud utilisé par la $n^{ième}$ tâche d'un groupe.

Les variables `VIGNE_GROUPE` et `VIGNE_GROUPE_TACHE_n` sont propres à un groupe et ne concernent qu'un sous-ensemble des tâches de l'application. Ces variables ont des valeurs différentes selon le groupe de tâches dans lequel elles sont définies.

Finalement, il est possible qu'à certaines étapes de l'exécution, toutes les variables `VIGNE_TACHE_n` ne soient pas fixées. C'est par exemple le cas avec des applications de type *workflow* lorsqu'une tâche B dépend du résultat d'une tâche A pour débuter son exécution. Dans ce cas, le nœud utilisé par la tâche B n'est pas encore alloué afin de ne pas gaspiller de ressources et donc il est inconnu.

Relations de précédence entre tâches Les applications ayant des tâches possédant des dépendances de précédence entre elles, comme les applications modélisées par un *workflow* nécessitent un support du système pour coordonner l'exécution des différentes tâches et le transfert des dépendances, typiquement des fichiers, entre les tâches.

Nous avons vu précédemment que l'allocation d'une ressource (éventuellement une portion des ressources physiques d'un nœud de grille) implique son verrouillage pour qu'elle ne puisse pas être utilisée par plusieurs applications afin de ne pas dégrader les performances d'exécution. Ainsi lorsque des groupes de tâches d'une application sont en attente du résultat d'autres groupes, c'est-à-dire lorsqu'il y a une dépendance de précédence entre ces groupes, les ressources pour les groupes en attente ne sont pas allouées.

Lorsque le système Vigne exécute une application possédant des dépendances de précédence, la gestion du cycle de vie de chaque tâche est modifiée. En effet, si le GCV d'une tâche détecte que cette tâche dépend du résultat d'autres tâches, le pilotage est suspendu dès le début du cycle de vie, avant même l'interrogation du système d'information.

L'algorithme 19 est exécuté lorsque le GCV commence le pilotage d'une tâche. Le GCV de la tâche commence par vérifier que toutes les tâches dont dépend la tâche concernée ont terminé leur exécution (cf. lignes 5 à 10). Si la condition est vérifiée, le GCV poursuit l'exécution de la tâche (cf. ligne 12), sinon il se met en attente. La procédure `Verification-`

Algorithme 19 Vérification des dépendances de précédence

```
 1: procedure VERIFICATIONDEPENDANCESPRECEDENCE(tache)
 2:     liste ← ObtenirListeDesTachesDependantes(tache)
 3:     dependancesPretes ← vrai
 4:     i ← 0
 5:     while (i < taille(liste) ∧ (dependancesPretes = faux) do
 6:         t ← liste[i]
 7:         if (t.etat ≠ EXECUTION_TERMINEE) then
 8:             tachesPretes ← faux
 9:         end if
10:     end while
11:     if dependancesPretes = vrai then
12:         DemanderPassageEtatAllocationDeRessources
13:     else
14:         /* nous ne faisons rien ici, il suffit d'attendre */
15:     end if
16: end procedure
```

`DependancesPrecedences()` est appelée par le GCV d'une tâche A lorsqu'un GCV responsable d'une autre tâche dont A attend le résultat termine la dernière étape du cycle de vie de sa tâche. Ainsi, si l'exécution d'une tâche A est conditionnée par le résultat de plusieurs tâches, `VerificationDependancesPrecedence(A)` est appelée plusieurs fois.

Les dépendances de précédence entre éléments applicatifs sont exprimées par l'utilisateur au moment de la soumission de l'application et décrites par $\mathscr{D}$. Nous définissons $\mathscr{D} = \bigcup_{i=1}^{k} Dep_{\mathscr{T}_i}$, avec k le nombre de tâches qui composent l'application, $Dep_{\mathscr{T}_i} = \emptyset \vee \{\{f_1, f_2, ...\}_{\mathscr{T}_x} \cup \{f_1, f_2, ...\}_{\mathscr{T}_y} \cup ...\}$. $\mathscr{T}_x, \mathscr{T}_y, etc$ sont les tâches dont dépend $\mathscr{T}_i$ et f_1, f_2, etc sont les fichiers produits par les tâches $\mathscr{T}_x, \mathscr{T}_y, etc$ qui sont utilisés par $\mathscr{T}_i$.

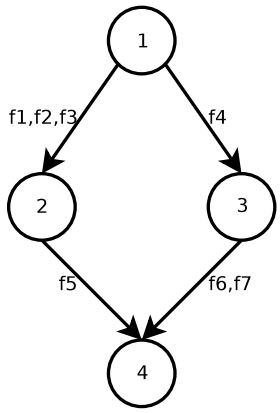


FIG. 5.8 – Exemple d'application *workflow* avec échange de fichiers entre les tâches

La figure 5.8 montre un exemple d'application de type *workflow* où des fichiers sont échangés entre les tâches. Tout comme pour les tâches ayant des dépendances de synchronisation et spatiales entre elles, c'est à l'utilisateur de décrire les dépendances de précédence entre tâches. Dans le formalisme présenté plus haut, l'application d'exemple de la figure 5.8 est dé-

crite ainsi : $\mathscr{D} = Dep_{\mathscr{T}_1} \cup Dep_{\mathscr{T}_2} \cup Dep_{\mathscr{T}_3} \cup Dep_{\mathscr{T}_4}$ avec $Dep_{\mathscr{T}_1} = \emptyset$, $Dep_{\mathscr{T}_2} = \{f1, f2, f3\}_{\mathscr{T}_1}$, $Dep_{\mathscr{T}_3} = \{f4\}_{\mathscr{T}_1}$ et $Dep_{\mathscr{T}_4} = \{f5\}_{\mathscr{T}_2} \cup \{f6, f7\}_{\mathscr{T}_3}$.

La fonctionnalité présentée dans ce paragraphe permet au système Vigne d'exécuter des applications de type *workflow* où les tâches possèdent des dépendances de précédence entre elles. Dans notre conception, nous avons choisi de n'allouer les ressources aux tâches que lorsqu'elles doivent être réellement exécutées afin de ne pas gaspiller de ressources en les réservant toutes dès le début. Cela peut poser un problème si une tâche dépendant du résultat d'une autre tâche nécessite une ressource qui n'existe pas dans la grille. Dans ce cas, le début de l'application aura été exécuté pour rien et l'utilisateur ne sera pas immédiatement prévenu de l'échec de l'exécution de son application. De même, si la tâche concernée nécessite une ressource rare, très occupée, l'application risque d'être bloquée longtemps avant de continuer son exécution. La réservation des ressources pourrait résoudre en partie le problème. Si des ressources nécessaires n'existaient pas, l'application ne serait pas exécutée. Malgré cela, il reste le cas pathologique où une ressource rare est réservée et subit une défaillance. L'application dans ce cas aura consommé des ressources dans la grille sans pouvoir arriver au terme de son exécution. De plus, deux types de réservation peuvent être faites. Une réservation immédiate de toutes les ressources nécessaires pour une durée indéterminée qui serait coûteuse en ressources et une réservation indépendante pour chaque tâche qui tient compte de leur durée d'exécution, qui cette fois économise les ressources mais qui est complexe à utiliser. Il n'est pas forcément aisé de connaître à l'avance le temps d'exécution des tâches de l'application. Notre stratégie d'allocation *à la demande* des ressources, comme nous venons de le voir, ne résout pas tous les problèmes mais elle convient dans de nombreux cas puisqu'elle offre de bons résultats en termes d'occupation des ressources et qu'elle est simple à utiliser.

5.1.3.3 Automate complet du gestionnaire de cycle de vie

La figure 5.9 présente l'automate complet du gestionnaire de cycle de vie qui prend en compte les applications multi-tâche. Nous rappelons qu'à chaque tâche correspond un GCV et donc un automate d'état. Comparativement à l'automate du GCV pour les applications mono-tâche de la figure 5.7, quatre états ont été ajoutés.

L'état E1 est inséré en tête de la chaîne pour attendre que toutes les autres tâches dont dépend la tâche ont terminé correctement leur exécution. E1 est donc utilisé pour traiter les dépendances de précédence.

L'état E4 a également été inséré pour traiter les dépendances de synchronisation entre tâches. Lorsque le GCV d'une tâche arrive dans cet état, il attend que tous les GCV des tâches appartenant au même groupe que lui aient terminé l'étape d'allocation de ressources.

L'état E6 est utilisé pour attendre le transfert des fichiers de dépendance depuis les tâches terminées vers le nœud de la tâche courante.

Finalement, l'état E10 est utilisé pour notifier les tâches qui dépendent de la tâche courante et qui sont en attente dans l'état E1.

Bien entendu, les états E1 et E6 sont ignorés si la tâche concernée ne dépend pas du résultat d'autres tâches. De même pour l'état E10 si aucune tâche ne dépend de cette tâche et pour l'état E4 si la tâche n'appartient pas à un groupe de tâches ayant une dépendance de synchronisation entre elles.

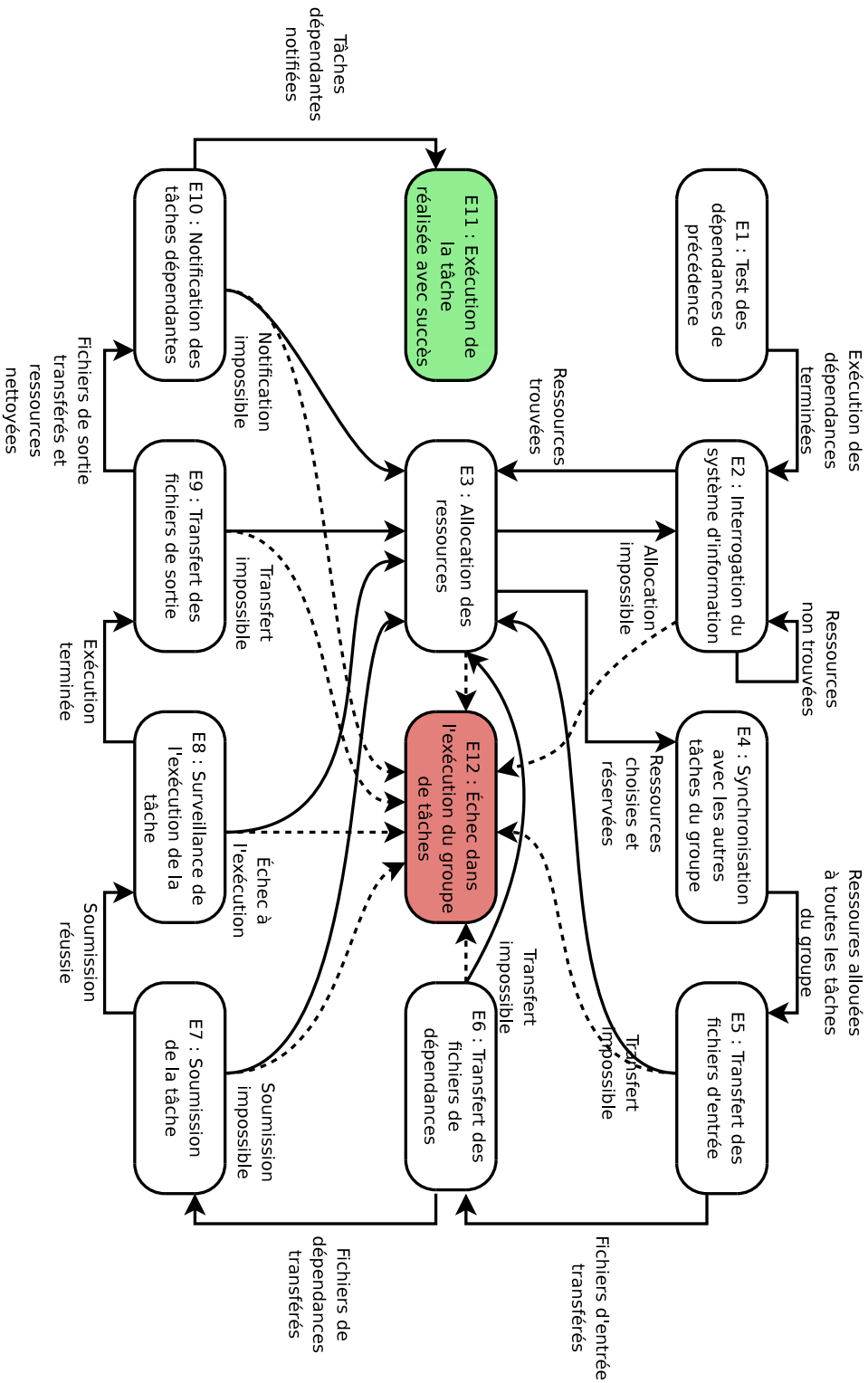


FIG. 5.9 – Automate d'état du gestionnaire de cycle de vie pour une application multi-tâche

Nous pouvons également remarquer sur cet automate qu'à partir de l'état E5, les erreurs peuvent être traitées de deux façons. En effet, si la tâche considérée n'appartient pas à un groupe de tâche ayant une dépendance de synchronisation alors son GCV retourne dans l'état E3, sinon il se met dans l'état E12 qui correspond à l'état d'erreur. Lorsqu'une tâche appartient à un groupe de tâches liées par une dépendance de synchronisation, son GCV ne peut plus revenir dans un état antérieur à E4 après la synchronisation du groupe de tâches. En effet, une fois cette barrière de synchronisation franchie, les tâches du groupe peuvent communiquer ensemble. Or pour cela, la localisation physique de nœuds hébergeant les tâches ne doit plus changer car les variables d'environnement `VIGNE_TACHE_n`, `VIGNE_GROUPE_TACHE_n` et `VIGNE_GROUPE` sont positionnées sur chaque nœud une fois pour toutes après la barrière de synchronisation. C'est grâce à ces variables d'environnement que les communications peuvent être initiées puisqu'elles contiennent la localisation physique des nœuds qui hébergent les tâches. Nous supposons que lorsqu'un flux de communication est établi entre deux tâches, ses extrémités ne peuvent pas être déplacées. En perspective de travail, il serait intéressant d'utiliser des mécanismes similaires à ceux proposés dans Kerrighed concernant les *flux dynamiques* [91] qui permettent de déplacer les extrémités des flux de communication pour autoriser le déplacement de tâches communicantes. En revanche, si la tâche courante n'est pas liée par une dépendance de synchronisation avec d'autres tâches, l'exécution de la tâche peut être rejouée en cas de problème, c'est-à-dire que l'allocation de nouvelles ressources peut avoir lieu.

Si un GCV arrive dans l'état E12 et que la tâche dont il pilote l'exécution appartient à un groupe de tâches liées par une dépendance de synchronisation, nous considérons que l'exécution de tout le groupe de tâches est compromis. En effet, des tâches du groupe pourraient vouloir communiquer avec la tâche ayant subit la défaillance. Dans ce cas, le service de gestion d'application doit interrompre tous les GCV responsables des tâches du groupe, nettoyer toutes les ressources utilisées par les exécutions en cours et les repositionner dans l'état E2. Une étude plus approfondie devrait être menée pour modifier le comportement du GCV dans ce cas si des applications tolérantes aux défaillances d'une partie des tâches qui les composent sont exécutées. En effet, il ne serait pas forcément nécessaire d'interrompre les autres tâches du groupe.

5.2 Superviser les applications pour fiabiliser leur exécution

Dans cette partie nous détaillons les fonctionnalités apportées par notre système pour garantir la fiabilité de l'exécution des applications. Les défaillances de nœuds ou d'applications étant inéluctables, nous présentons un composant du service de gestion d'application, nommé superviseur d'application, qui est chargé de détecter les défaillances des ressources utilisées pour une exécution et les défaillances de tâches. Ce travail a été, pour une partie, réalisé en collaboration avec Thomas Ropars lors de son stage[1] de Master de recherche [148].

[1] Stage dont j'ai assuré le co-encadrement

5.2.1 Architecture du superviseur d'application

Le superviseur d'application est constitué de deux types de composants. Le premier type, nommé superviseur de tâche, est chargé de surveiller les processus système des tâches qui sont en cours d'exécution. Le second type, nommé superviseur d'application, est chargé de surveiller l'ensemble des superviseurs de tâches d'une application.

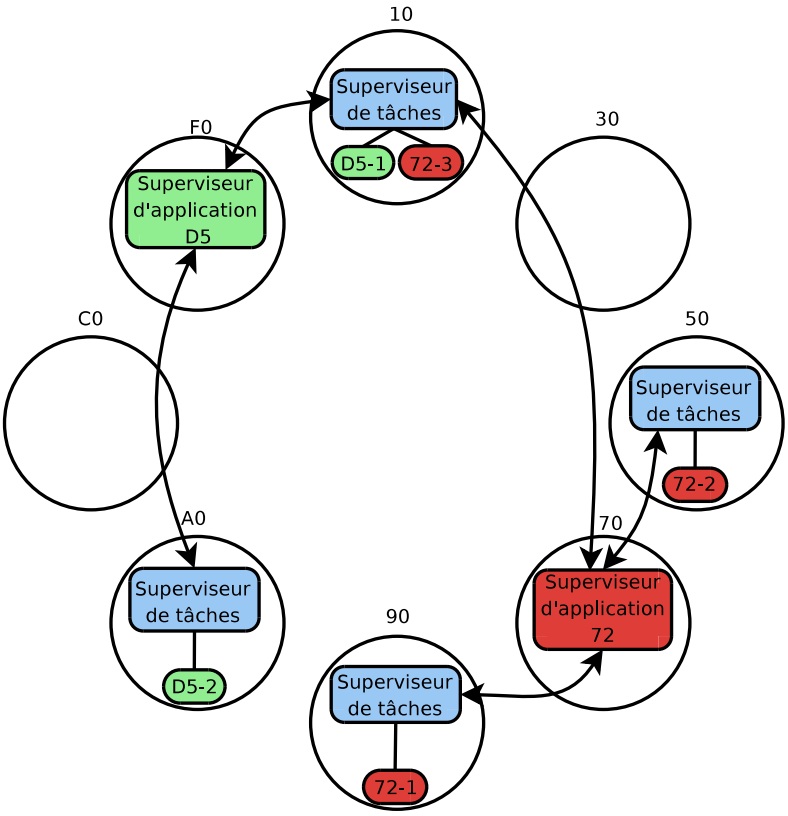


FIG. 5.10 – Exemple de grille avec deux superviseurs d'application

La figure 5.10 présente un exemple de grille composée de huit nœuds où deux applications sont exécutées. L'application possédant l'identifiant `72` est composée de trois tâches nommées `72-1`, `72-2` et `72-3`. L'application possédant l'identifiant `D5` est composée de deux tâches nommées `D5-1` et `D5-2`. Il y a un superviseur d'application par application qui est intégré au service de gestion d'application. Il est situé sur le nœud ayant l'identifiant numérique le plus proche de l'identifiant de l'application, c'est-à-dire sur le nœud du gestionnaire d'application. Sur chaque nœud où au moins une tâche est exécutée, un superviseur de tâches est déployé. Un superviseur de tâches peut superviser des tâches de plusieurs applications si elles sont exécutées sur le même nœud comme par exemple sur le nœud 10.

5.2.2 Superviseur de tâches

Le superviseur de tâches permet de surveiller finement l'exécution des tâches s'exécutant sur un nœud de la grille. Il permet en particulier de suivre le cycle d'exécution des tâches et de contrôler les ressources qu'elles utilisent.

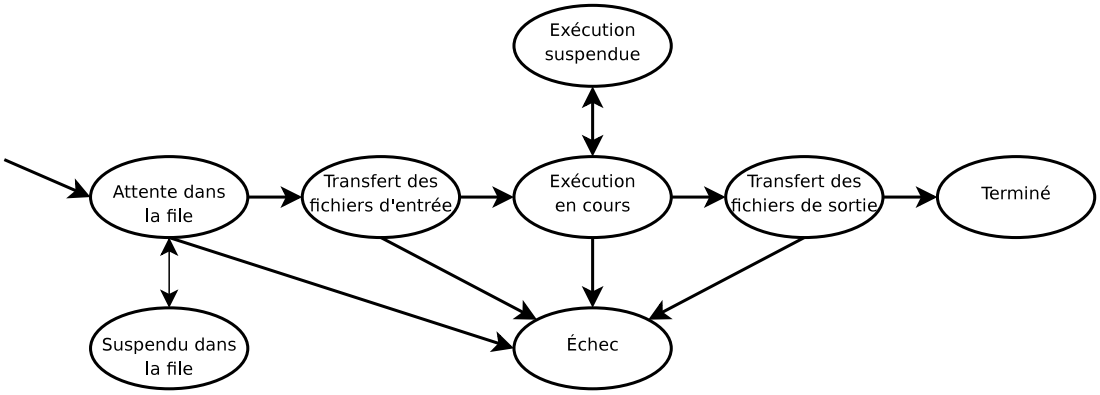


FIG. 5.11 – Cycle d'exécution d'une tâche exécutée sur une ressource exploitée par un gestionnaire de traitement par lots

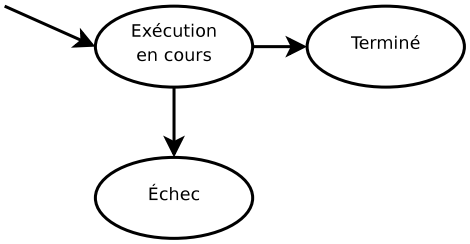


FIG. 5.12 – Cycle d'exécution d'une tâche exécutée sur une ressource exploitée en mode interactif

Selon le type de ressource, le cycle d'exécution d'une tâche est différent. La figure 5.11 présente le cycle d'exécution d'une tâche lancée sur une ressource exploitée à l'aide d'un gestionnaire de traitement par lots tel que cela est défini par la norme DRMAA (*Distributed Resource Management Application API*) [8]. Dans le cas des ressources exploitées en mode interactif, le cycle d'exécution est beaucoup plus simple, comme le montre la figure 5.12, puisque seuls les états *exécution en cours*, *terminé* et *échec* sont présents.

Quel que soit le mode d'exploitation des ressources, le superviseur de tâches reporte au superviseur d'application les changements d'état majeurs dans le cycle d'exécution des tâches. Ces changements majeurs correspondent aux états présents dans le cycle d'exécution des tâches sur des ressources exploitées en mode interactif. En effet, ce n'est que dans ces états que la gestion d'application est modifiée.

En mode non-interactif, c'est-à-dire lorsque les ressources sont exploitées avec un gestionnaire de traitement par lots, le superviseur de tâches communique avec le gestionnaire de

traitement par lots. Selon le gestionnaire de traitement par lots, les informations obtenues sur l'exécution sont plus ou moins précises mais permettent d'obtenir dans tous les cas la consommation mémoire, la consommation processeur et le statut de la tâche en cas de défaillance.

En mode interactif, le superviseur de tâches communique directement avec le système d'exploitation et permet d'obtenir des informations très précises sur la consommation des ressources et sur les raisons qui auraient conduit à une défaillance de l'exécution d'une tâche.

Grâce au superviseur de tâches, le système de grille peut détecter au plus tôt les défaillances des tâches et les reporter au superviseur d'application. Les défaillances de tâches peuvent être liées au contexte d'exécution ou au code de l'application.

Si la défaillance est liée au contexte d'exécution, comme par exemple dans le cas où l'espace de disque temporaire devient nul, le système pourra, à l'aide des informations reportées, prendre une décision pour éventuellement exécuter la tâche qui a subi une défaillance sur une autre ressource.

Dans le cas où la défaillance est liée à une erreur dans le code de l'application, par exemple une application qui voudrait accéder à une zone de mémoire non allouée, les informations reportées peuvent aider l'utilisateur à comprendre la raison de la défaillance. En effet, le superviseur de tâches peut reporter au système qu'une tâche s'est terminée suite à la réception d'un signal, par exemple, le signal `SIGSEGV` dans le cas d'une tentative d'accès à une zone de mémoire non allouée.

Les informations de supervision collectées par un superviseur de tâches concernant notamment les ressources consommées sont stockées sur le nœud hébergeant le superviseur de tâches considéré et sont synthétisées puis envoyées au superviseur d'application en fin d'exécution.

5.2.3 Superviseur d'application

Le superviseur d'application est chargé de synthétiser les informations envoyées par tous les superviseurs de tâches localisés sur les mêmes nœuds que les tâches appartenant à l'application qu'il supervise.

5.2.3.1 Modèle de communication

Nous avons vu dans la partie 5.2.2 que le superviseur d'application recevait des notifications de la part des superviseurs de tâche. Ce mode de communication permet de minimiser les communications entre les superviseurs de tâches et le superviseur d'application puisque les communications n'ont lieu que lorsque cela est nécessaire, c'est-à-dire lors d'un changement d'état de la tâche qui modifie le cycle de vie de l'application.

Le superviseur d'application peut tout de même contacter les superviseurs de tâches lorsqu'un utilisateur effectue une demande d'information sur l'exécution d'une application. Dans ce cas, le superviseur de l'application concernée interroge tous les superviseurs de tâches impliqués dans l'exécution pour obtenir l'état de toutes les tâches de l'application. Puis, le superviseur d'application retourne à l'utilisateur les informations collectées.

Les communications dans le sens superviseur d'application vers superviseur de tâches sont réalisées en mode point-à-point puisque les superviseurs de tâches ne peuvent pas être déplacés sans que le superviseur d'application ne soit averti. Dans l'autre sens, les communications sont

réalisées par routage, car le gestionnaire d'application peut être déplacé en cas de défaillance ou d'ajout d'un nœud de la grille.

5.2.3.2 Détection des ressources défaillantes

Le rôle du superviseur d'application est également de s'assurer que les ressources utilisées par les tâches en exécution de l'application qu'il supervise ne sont pas défaillantes. Un nœud défaillant est : soit un nœud qui connaît une défaillance matérielle et donc qui entraîne l'interruption de son fonctionnement, soit un nœud qui se retrouve isolé suite à la défaillance d'un lien réseau. Pour cela, il envoie un *ping* à un intervalle régulier de γ secondes à tous les superviseurs de tâches hébergeant des tâches de l'application qu'il supervise. Si un superviseur de tâches ne renvoie pas de *pong*, alors la tâche supervisée est supposée défaillante. Si des communications ont lieu entre le superviseur d'application et un superviseur de tâches, comme c'est le cas lorsqu'un superviseur de tâches effectue une notification de changement d'état d'une tâche qu'il supervise ou lorsqu'un utilisateur a demandé des informations sur l'exécution d'une application, le *ping* suivant est décalé de γ secondes car il est évident dans ce cas que le superviseur de tâches est toujours accessible, et donc que le nœud n'est pas défaillant.

Lorsqu'un superviseur d'application détecte la défaillance d'un nœud utilisé par une tâche applicative, il exclut ce nœud de l'exécution et notifie la défaillance au GCV de la tâche qui peut réagir comme cela est expliqué dans la partie 5.2.4. Si la défaillance est temporaire, par exemple si un lien réseau est momentanément coupé, et que ce nœud redevient accessible, il est ignoré par le superviseur d'application qui ordonne dans ce cas l'arrêt de l'exécution des tâches en cours sur ce nœud. Les nœuds sont donc évincés dès leur première défaillance, ce qui garantit que les exécutions ne sont pas perturbées par le comportement aléatoire de nœuds instables. Si la défaillance concerne une tâche appartenant à un groupe de tâches ayant une dépendance de synchronisation, toutes les tâches du groupe sont arrêtées et une décision de ré-ordonnancement peut être prise pour chaque tâche. Dans le cas contraire, seulement une décision concernant la tâche victime de la défaillance est prise.

5.2.4 Réaction à une défaillance de tâche

Qu'un superviseur de tâches détecte une défaillance dans l'exécution d'une tâche ou qu'un superviseur d'application détecte la disparition d'un nœud hébergeant une tâche applicative, le superviseur d'application reporte la défaillance au gestionnaire de cycle de vie de la tâche concernée. Nous examinons dans les paragraphes suivants les alternatives possibles lorsqu'une telle défaillance est reportée.

5.2.4.1 Redémarrage d'une tâche depuis le début

Nous avons vu dans la partie 5.1.2 sur la gestion d'applications mono-tâche que lorsqu'une défaillance intervient lors de l'exécution, le GCV peut ré-exécuter cette tâche. Cette ré-exécution peut être réalisée plusieurs fois. Cependant, si la cause de la défaillance est clairement identifiée comme étant liée au code de l'application et que la même erreur est générée à la suite de multiples exécutions, par exemple si la tâche est systématiquement tuée suite à

la réception du signal `SIGSEGV`, alors la tâche n'est pas ré-exécutée. En revanche, si les défaillances sont dues à une perte de nœuds, par exemple suite à la rupture de liens réseaux, le système essaye d'exécuter la tâche tant que des ressources sont libres.

En revanche, dans le cas d'une application multi-tâche (cf. partie 5.1.3), la ré-exécution d'une tâche défaillante ne suffit pas forcément. En effet, si la défaillance touche une tâche appartenant à un groupe de tâches liées par une dépendance de synchronisation, alors tout le groupe de tâches doit être ré-exécuté. Si le système détecte que la défaillance d'une tâche est imputable au code de l'application, alors toute l'application doit être arrêtée.

La stratégie consistant à redémarrer les tâches ou les groupes de tâches défaillantes depuis le début a l'avantage de pouvoir être utilisée quel que soit le type de ressource utilisé. Toutefois, l'inconvénient majeur de cette solution est que les ressources consommées jusqu'à la défaillance sont gaspillées.

5.2.4.2 Redémarrage d'une tâche depuis un point de reprise

Certains nœuds peuvent disposer de mécanismes permettant de prendre des points de reprise des applications comme c'est le cas par exemple des nœuds exploités avec le système à image unique Kerrighed [124] ou avec un système Linux étendu par BLCR (Berkeley Lab Checkpoint/Restart) [4]. Dans ce cas, il serait intéressant que le superviseur de tâches prenne des points de reprise des tâches en exécution et qu'il les envoie sur un serveur de fichiers.

En cas de défaillance des nœuds ou en cas de défaillance des tâches provoquée par le contexte d'exécution, les tâches pourraient être redémarrées depuis le dernier point de reprise et ainsi éviter le gaspillage de ressource engendré par un redémarrage depuis le début. Ce mécanisme est simple à mettre en œuvre pour les applications mono-tâche et les tâches indépendantes des applications multi-tâche.

Pour les tâches appartenant à un groupe de tâches possédant une dépendance de synchronisation entre elles, le mécanisme de point de reprise doit prendre en compte les communications entre les tâches du groupe pour qu'un état global cohérent soit obtenu à l'issue d'un redémarrage. Dans l'état de l'art, un support générique pour prendre un point de reprise d'une application possédant des tâches communicantes n'est que très peu traité. Le système Kerrighed propose tout de même une approche limitée dans [79] et cette problématique est actuellement à l'étude dans le projet XtreemOS [36]. D'autres travaux, cette fois non génériques, s'intéressent à la tolérance aux défaillances d'applications écrites dans le paradigme MPI comme par exemple le projet MPICH-V [58]. Cette approche requiert la ré-édition des liens des applications avec la librairie MPICH-V et le déploiement d'une infrastructure spécifique dans la grille comme des serveurs destinés à accueillir les points de reprise.

En conclusion, les mécanismes transparents pour les applications permettant de prendre des points de reprise d'applications composées de tâches communicantes sont encore à l'étude et non fonctionnels. Souhaitant supporter les applications patrimoniales, nous ne pouvons pas retenir une stratégie où les tâches défaillantes sont redémarrées depuis un point de reprise si elle n'est pas complètement transparente pour les applications communicantes. Pour résoudre ce problème, la solution réside probablement dans la modification du noyau du système d'exploitation utilisé sur les ressources de la grille afin d'assurer la transparence et la généricité d'utilisation. L'approche suivie par Kerrighed [79] qui étend le noyau Linux n'est pas suffi-

sante à l'échelle d'une grille puisque des tâches communicantes peuvent être exécutées sur différentes grappes. Dans ce cas, les systèmes des grappes concernées devraient être coordonnés pour assurer qu'un état global cohérent des tâches soit enregistré dans les points de reprise. Ainsi, le projet XtreemOS [36] qui s'autorise la modification du système d'exploitation et dont l'architecture remonte jusqu'au niveau grille semble le plus prometteur pour traiter ce problème.

Certaines applications intègrent un mécanisme de point de reprise. Il serait intéressant de proposer aux utilisateurs la possibilité de décrire les capacités des applications de ce point de vue afin de les exploiter dans le service de gestion d'application en cas de défaillance.

5.3 Conclusion

Dans ce chapitre, nous avons présenté une approche pour exécuter de façon simple, efficace et fiable des applications sur une grille.

Plus particulièrement, nous avons détaillé le service de gestion d'application de notre système. Ce service repose sur des fondations pair-à-pair qui lui permettent d'être complètement distribué et qui lui confèrent une excellente fiabilité dans une grille dont les nœuds sont volatils. Le service de gestion d'application permet de prendre en charge divers types d'applications telles que les applications parallèles ou encore les applications de type *workflow*. En effet, le service de gestion d'application est capable de prendre en compte différentes dépendances entre les tâches d'une application. Ces dépendances peuvent être spatiales, de synchronisation ou de précédence et elles sont exprimées dans un formalisme simple par l'utilisateur lorsqu'il soumet une application. Notre service intègre un moteur de pilotage des tâches pour enchaîner des tâches possédant des dépendances de précédence entre elles. Cela permet de simplifier l'exécution de ce type d'application car l'enchaînement est automatique. De plus, les échanges de fichiers entre ces différentes tâches sont également réalisés automatiquement par le système afin de donner aux utilisateurs l'illusion que le système met en œuvre un système de fichier distribué à l'échelle de la grille. Doté de mécanismes permettant de modifier l'environnement d'exécution des tâches, les applications multi-tâche comme les applications parallèles peuvent être exécutées sans modifications du code applicatif. Le service de gestion d'application repose sur le service d'allocation des ressources permettant de co-allouer des ressources proches du point de vue du réseau. Cela permet ainsi aux tâches fortement communicantes des applications d'être exécutées sur des ressources permettant des communications réseau à faible latence lorsque cela est nécessaire pour une exécution efficace. Dans l'état de l'art, divers intergiciels supportent l'exécution d'applications distribuées. C'est par exemple le cas de Globus munit de DUROC [70], KOALA [123], ProActive [47] ou encore Vishwa [168]. Cependant aucun ne fournit la transparence nécessaire à l'exécution d'applications patrimoniales.

Nous avons également détaillé les mécanismes de supervision d'applications de notre système. Ces mécanismes permettent de détecter les défaillances de ressources et les défaillances des applications afin de prendre des mesures pour mener malgré tout l'exécution des applications à leur terme lorsque c'est possible et dans les meilleurs délais. Le superviseur d'application permet de fiabiliser l'exécution des applications en détectant au plus tôt les défaillances. Il peut aussi aider les développeurs d'applications en leur reportant des informations sur les

terminaisons anormales qui ne sont pas liées au contexte d'exécution.

Pour conclure, nous avons présenté un service de gestion d'application permettant d'exécuter simplement et efficacement une large gamme d'applications. Complété par un superviseur d'application, le service de gestion d'application permet d'exécuter ces applications de façon fiable puisque les gestionnaires d'application sont fiables et puisque le superviseur d'application permet de détecter les défaillances et de notifier les GCV pour qu'ils redémarrent les tâches en cas de problème.

Chapitre 6

Éléments de mise en œuvre et évaluation du système Vigne

Afin de valider les concepts présentés dans les chapitres 4 et 5, nous avons réalisé un prototype. Dans ce chapitre, nous présentons quelques éléments de mise œuvre du prototype et une évaluation couvrant les contributions que nous avons apportées à l'état de l'art.

6.1 Quelques éléments de mise en œuvre

Nous présentons dans cette partie quelques éléments de mise en œuvre. Tout d'abord nous décrivons globalement le prototype que nous avons développé, puis nous détaillons deux points de mise en œuvre.

6.1.1 Prototype

Nous avons mis en œuvre les concepts présentés dans cette thèse en étendant le prototype Vigne, initialement développé par Louis Rilling lors de sa thèse [143].

6.1.1.1 Services mis en œuvre

La figure 6.1 présente l'architecture globale que nous avons proposée dans la partie 3.3 ainsi que les services qui ont été mis en œuvre.

Nous avons donc mis en œuvre les services suivants :

- interface avec les ressources pour les ressources exploitées en mode interactif (interface avec le *shell*) et les ressources exploitées avec le gestionnaire de travaux Torque ;
- service de communication sur réseau non-structuré ;
- système d'information ;
- service d'allocation de ressources ;
- service de transfert de fichiers ;
- service de gestion d'application ;

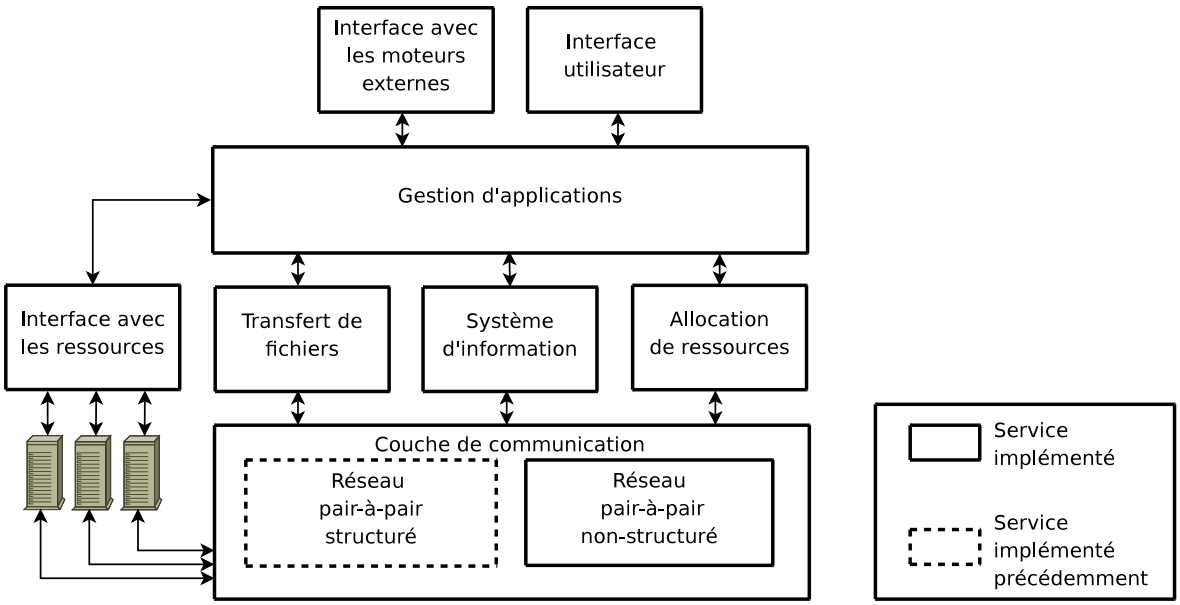


FIG. 6.1 – Architecture et services mis en œuvre dans Vigne

– interface utilisateur ;
– interface avec les moteurs externes de pilotage d'applications.

Le service de communication sur réseau structuré et en mode point-à-point, largement utilisé pour nos travaux, a été repris de l'implémentation réalisée par Louis Rilling. Une infrastructure permettant d'exécuter des actions de façon différée (*timers*) a également été reprise de l'implémentation de Louis Rilling. Cette infrastructure est utilisée dans tous les services que nous avons mis en œuvre.

6.1.1.2 Utilisation

Le prototype Vigne est utilisable à la fois en conditions réelles d'exécution, c'est-à-dire qu'il permet l'exécution d'applications réelles sur des ressources distribuées, et dans un mode de simulation pour évaluer certaines propriétés du système sans avoir à réaliser un déploiement sur des ressources réelles et en garantissant une parfaite reproductibilité de la grille simulée pour ce qui concerne le nombre de nœuds et la latence entre les nœuds.

Utilisation réelle Dans ce cas, le prototype est matérialisé par un démon, nommé `vigne_d`, et un client, nommé `vigne_c`.

Du point de vue de l'administrateur, une instance de `vigne_d` est déployée sur chaque ressource de la grille. Une ressource peut être un calculateur exploité de façon interactive, une grappe exploitée par un SIU comme Kerrighed ou une grappe exploitée par un gestionnaire de traitement par lots comme Torque. Si la ressource est un calculateur, `vigne_d` est directement exécuté au-dessus. Si la ressource est une grappe exploitée par un SIU, `vigne_d` est exécuté sur n'importe quelle calculateur de la grappe puisque l'ensemble des ressources de la grappe

```
<resource_description>
  <cpu_architecture>X86-64</cpu_architecture>
  <nb_nodes>1</nb_nodes>
  <os>LINUX</os>
  <os_version>2.6.21</os_version>
  <cluster_scheduler>SH</cluster_scheduler>
  <network>MYRINET</network>
  <memory>2048</memory>
</resource_description>
```

FIG. 6.2 – Exemple de fichier pseudo-XML décrivant les ressources d'un nœud Vigne

est virtuellement accessible sur chaque calculateur. Si la ressource est une grappe exploitée par un gestionnaire de traitement par lots, `vigne_d` est exécuté sur le nœud frontal de la grappe. À l'exécution de `vigne_d`, le fournisseur de ressources doit fournir un fichier décrivant les ressources dans un format pseudo-XML. La figure 6.2 montre un exemple de description pour un nœud exploité en mode interactif et possédant un processeur d'architecture x86-64, 2 Go de mémoire vive, une carte réseau de type Myrinet et un système Linux doté d'un noyau 2.6.21.

Du point de vue de l'utilisateur, `vigne_c` permet d'interagir avec les nœuds exécutant le système Vigne (`vigne_d`). Deux types d'opérations peuvent être réalisées avec `vigne_c`. Le premier type concerne les opérations qui agissent sur les applications alors que le second type concerne les opérations qui agissent sur les ressources. D'une façon simplifiée, `vigne_c` s'utilise de la façon suivante :

```
>> vigne_c -h $HOSTNAME -p $PORT -q $QUERY_TYPE -a $ARGS
```

L'invocation du client Vigne nécessite de fournir le nom (`$HOSTNAME`) et le port (`$PORT`) d'une ressource Vigne, le type d'opération souhaitée (`$QUERY_TYPE`) et un ensemble d'arguments (`$ARGS`) dont le nombre et le type dépendent du type d'opération.

Si l'utilisateur souhaite effectuer une opération concernant une application, n'importe quelle ressource peut être contactée pour soumettre l'opération. Cela est rendu possible grâce à la vision SIU offerte par Vigne. En effet, `vigne_c` envoie la requête à la ressource décrite par `$HOSTNAME` et `$PORT`. Cette ressource est chargée d'encapsuler la requête dans un message Vigne qui est automatiquement routé sur le nœud exécutant le gestionnaire d'application de l'application concernée par la requête.

Si l'utilisateur souhaite effectuer une opération sur une ressource, `$HOSTNAME` et `$PORT` doivent désigner explicitement cette ressource.

Le tableau 6.1 décrit les différentes valeurs que peut prendre `$QUERY_TYPE` ainsi que leur description.

Simulateur Le prototype Vigne développé initialement par Louis Rilling comporte deux modes de fonctionnement : le mode d'exécution réelle et le mode de simulation. En mode simulation, la couche de communication est modifiée pour que les messages soient traités par un moteur d'événements discrets. Les latences entre les nœuds sont constantes et le temps de traitement des opérations du système (en temps virtuel de simulation) est nul. Cela modélise

Cible	Type de requête	Description de l'action
Applications	SUBMIT_JOB	Soumettre une application.
	GET_JOB_STATE	Obtenir l'état d'une application.
	KILL_JOB	Arrêter l'exécution d'une application et nettoyer les ressources.
	EXTEND_JOB	Ajouter des tâches à une application.
	GET_OUTPUT_FILES	Récupérer les fichiers de sortie d'une application.
	TRANSFER_FILES	Transférer des fichiers entres deux tâches d'une application.
	UNBLOCK_COMPONENT	Envoyer une notification externe pour débloquer une tâche en attente d'une application.
Ressources	GET_LOAD	Obtenir la charge d'une ressource.
	CHANGE_LOG_DIR	Modifier le répertoire de stockage des fichiers journaux.
	USE_RW_CACHE	Utiliser des caches pour traiter les requêtes de marches aléatoires.
	SET_RW_CACHE_POLICY	Définir la politique de gestion de cache de requêtes de marches aléatoires.
	USE_ADVANCED_MONITORING	Utiliser les mécanismes de surveillance fine des tâches.

TAB. 6.1 – Différentes requêtes pouvant être effectuées avec le client Vigne

une grille où les actions réalisées par le système sont infimes par rapport au coût de communication entre deux nœuds, ce qui est le cas pour un bon nombre de grilles expérimentales comme Grid'5000. Compte-tenu de l'abstraction de la couche de communication, les services reposant dessus ne nécessitent que de faibles modifications pour fonctionner en mode simulation.

Nous avons étendu ce mode de simulation pour les services que nous avons mis en œuvre. En particulier, nous avons modifié l'interface d'accès aux ressources. En effet, en mode simulation, nous n'exécutons pas d'applications réelles mais des applications synthétiques et les ressources sont émulées. Soit UTV une unité de temps virtuel, les caractéristiques d'un nœud se résument à :

- son type $Type_{ressource}$ dont la valeur appartient à $\mathbb{N}$,
- sa capacité de calcul (cf. 6.1).

$$Capacité\_calcul = \frac{Nb_{instructions}}{UTV} \tag{6.1}$$

Une application est simplement définie par le type de ressource $Type_{ressource}$ qu'elle requiert et le nombre d'instructions qu'elle comporte $Nb_{instructions}$.

Le mode de simulation permet également l'exécution de plusieurs tâches en concurrence sur une ressource. À chaque fois qu'une tâche est lancée sur une ressource, la date de terminaison de toutes les tâches est ré-évaluée. Soit $\mathscr{T}$ l'ensemble des tâches en cours d'exécution sur la ressource et $\mathscr{T}_i \in \mathscr{T}$, alors le temps de calcul restant de chaque tâche est calculé par :

$$Temps\_virtuel\_restant(\mathscr{T}_i) = \frac{\left(Nb^{\mathscr{T}_i}_{instructions\_total} - Nb^{\mathscr{T}_i}_{instructions\_calculés}\right)}{Capacité\_calcul} \times |\mathscr{T}| \tag{6.2}$$

De même, lorsqu'une tâche se termine, les temps de terminaison de toutes les tâches en exécution sur la ressource sont recalculés avec la fonction définie dans 6.2.

En mode simulation, il n'y a pas d'échange de fichiers entre les nœuds, ni d'interface externe. Les opérations à réaliser comme l'ajout d'un nœud dans le système ou la soumission d'une application sont définies dans une trace qui est injectée dans le simulateur.

6.1.1.3 Code développé

Le prototype Vigne avait été développé par Louis Rilling en langage C. Ainsi, l'extension que nous y avons apportée a également été développée en langage C. Cette extension représente environ 15000 lignes de code sur les 30000 lignes au total que compte le prototype.

Le code source peut être compilé sur un système Linux et une machine dotée d'un processeur d'architecture i386 ou x86-64.

Le code est lié à trois librairies externes qui sont `glib2` (gestion des listes et des tables de hachage), `gcrypt11` (fonctions de hachage MD5 et SHA-1), et `xml2` (analyse syntaxique des fichiers XML).

D'un point de vue système, le système Vigne s'exécute en espace utilisateur, sans privilèges spéciaux. Cela permet de limiter l'inquiétude que pourrait avoir un fournisseur de ressources à installer Vigne sur les ressources qu'il met à disposition dans la grille.

6.1.2 Détails de mise en œuvre

Nous présentons deux détails techniques de mise en œuvre de notre protoype dans le système Vigne.

Dans un premier temps nous présentons le comportement du client Vigne par rapport au type d'opération que l'utilisateur effectue. En effet, certaines opérations sont bloquantes et l'utilisateur doit attendre qu'elles se terminent et d'autres ne le sont pas et peuvent s'exécuter en arrière plan.

Ensuite, nous expliquons la méthode que nous avons utilisée pour mettre en œuvre le superviseur de tâche présenté dans la partie 5.2 sur des ressources exploitées en mode interactif.

6.1.2.1 Schémas de communication des requêtes client

Nous avons vu dans la partie 6.1.1.2 qu'il était possible d'effectuer deux types de requêtes avec le système Vigne : celles dont la cible est une application et celles dont la cible est une ressource.

Dans le cas où la cible est une ressource, le schéma de communication est simple puisque `vigne_c` envoie ou récupère directement une donnée vers un démon `vigne_d`. La requête `GET_LOAD` est synchrone puisque la valeur de charge est attendue en retour et les requêtes `CHANGE_LOG_DIR`, `USE_RW_CACHE`, `SET_RW_CACHE_POLICY` et `USE_ADVANCED_MONITORING` sont asynchrones.

Dans le cas où la cible est une application, une indirection est ajoutée dans le schéma de communication puisque l'utilisateur envoie sa requête à n'importe quel nœud du système et que cette requête est routée vers le gestionnaire d'application avant d'être traitée. Les requêtes soumises suivent un des trois types de communication que nous présentons ci-dessous.

La figure 6.3 montre le premier type de schéma de communication pour une requête de l'utilisateur. Avec ce type de communication, la requête soumise induit une action par le gestionnaire d'application et éventuellement sur les ressources mais un résultat est directement retourné à l'utilisateur, sans attendre la fin de l'action. C'est le cas des requêtes `SUBMIT_JOB`, `EXTEND_JOB` et `KILL_JOB`. En effet, lorsque l'utilisateur soumet ou étend une application, le système lui retourne l'identifiant de l'application et la liste des tâches mais il n'attend pas la fin de l'exécution. De même, lorsqu'un utilisateur souhaite arrêter une tâche, l'opération de nettoyage des ressources utilisées peut être effectuée en arrière-plan sans que la fin de l'opération ne soit requise pour redonner la main à l'utilisateur. Ce type de requête est donc asynchrone.

La figure 6.4 montre le second schéma de communication pour une requête de l'utilisateur. Dans ce cas, la requête soumise induit une action par le gestionnaire d'application sur les ressources et la fin de l'opération doit être attendue pour retourner un résultat à l'utilisateur. C'est le cas des requêtes `GET_JOB_STATE`, `TRANSFER_FILES` et `UNBLOCK_COMPONENT`. Il est évident que pour obtenir l'état d'une tâche il faille attendre la fin de l'interrogation effectuée par le gestionnaire d'application. Les deux autres requêtes concernent le pilotage d'une application depuis un moteur externe de pilotage. Vigne doit donc garantir que les opérations ont été effectuées lorsque la main est retournée au moteur de pilotage externe. Par exemple, dans le cas où une tâche attend le résultat d'une autre tâche pour démarrer, le système doit garantir que les fichiers de résultats ont bien été transférés. Ce type de requête est synchrone.

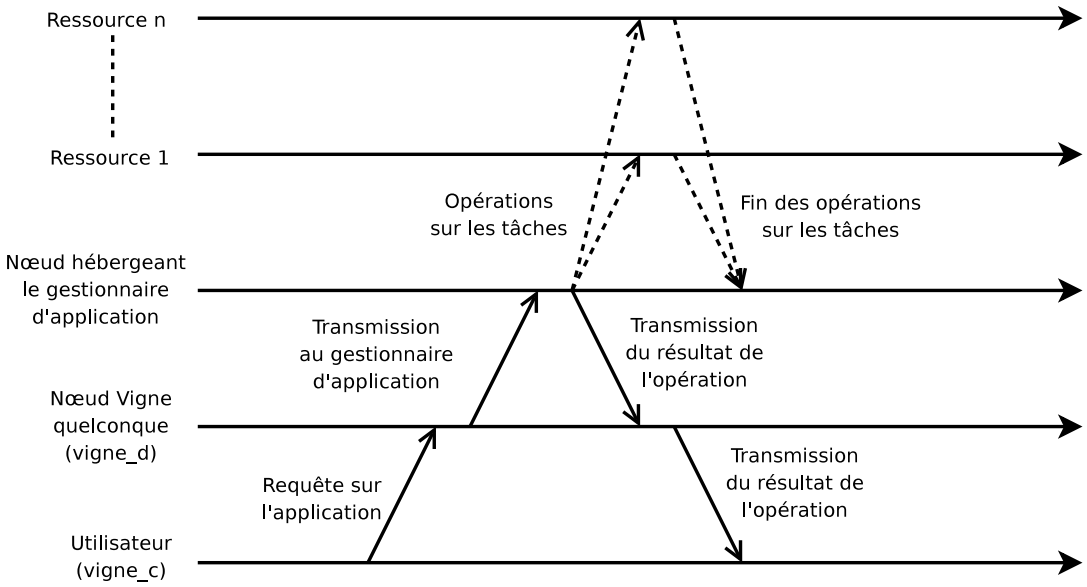


FIG. 6.3 – Schéma de communication pour les requêtes émises par `client_c` de type 1

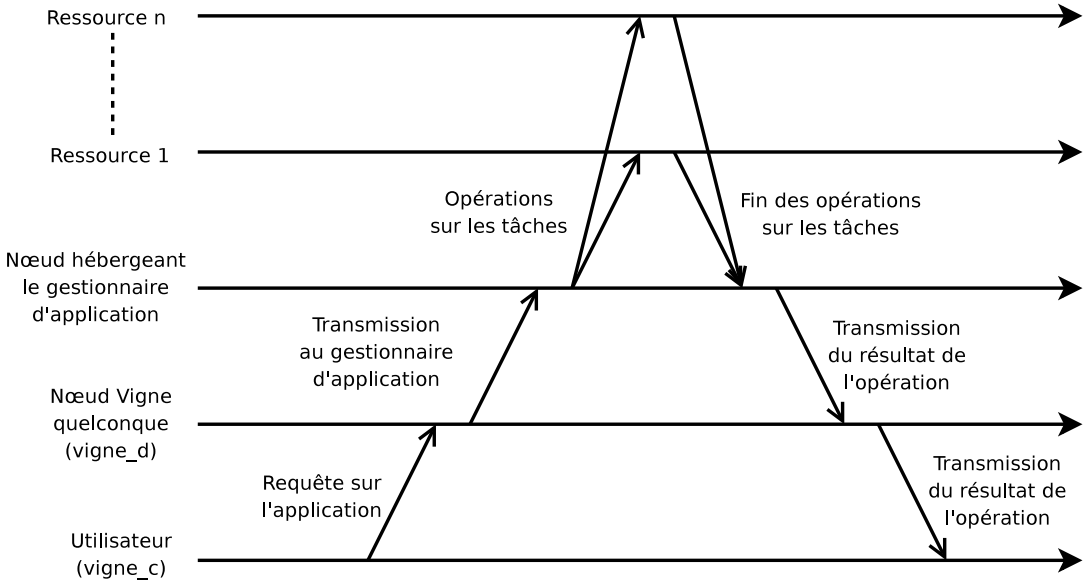


FIG. 6.4 – Schéma de communication pour les requêtes émises par `client_c` de type 2

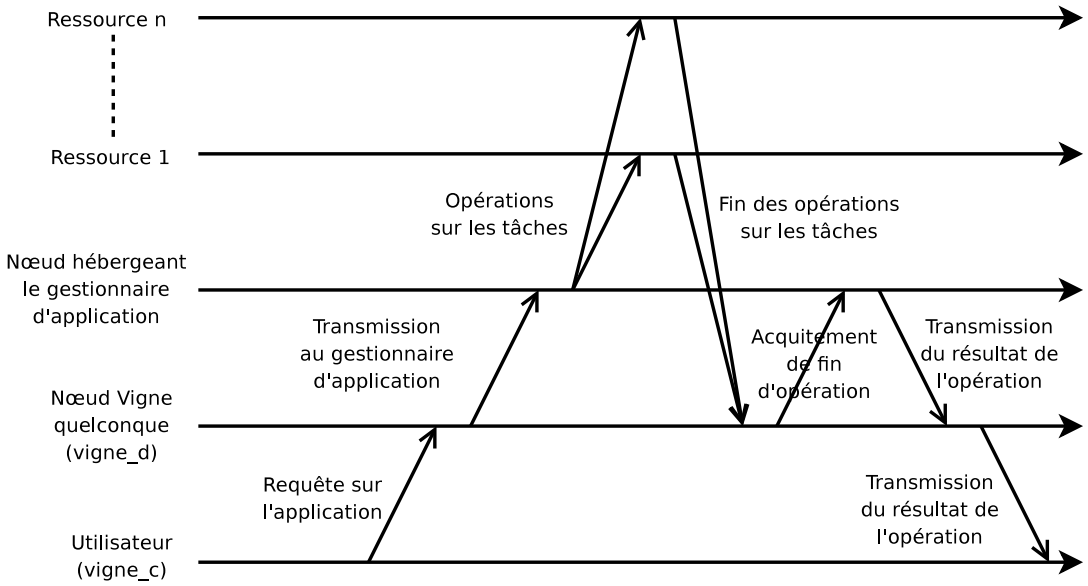


FIG. 6.5 – Schéma de communication pour les requêtes émises par `client_c` de type 3

La figure 6.5 montre le troisième schéma de communication pour la requête `GET_OUTPUT_FILES`. Dans ce schéma de communication, le gestionnaire d'application demande aux ressources d'effectuer un transfert direct d'un ensemble de fichiers sur un nœud choisi par l'utilisateur. Une fois le transfert terminé, le nœud récepteur avertit le gestionnaire d'application. La synchronisation avec le gestionnaire d'application est nécessaire car des fichiers de sortie ne peuvent être transférés qu'une fois car ils sont effacés en fin de transfert. Une fois que le gestionnaire d'application a reçu l'avertissement de bonne terminaison du transfert, il rend finalement la main à l'utilisateur. Ce type de requête est également synchrone.

6.1.2.2 Surveillance de l'exécution des tâches sur les ressources exploitées en mode interactif

Le service de gestion d'application est capable de réagir à la défaillance de tâches qui composent une application pour mener à bien l'exécution de cette application. Pour cela, le service de gestion d'application repose sur un mécanisme de détection de défaillance de tâche implémenté dans le superviseur de tâche (cf. partie 5.2.2) qui dépend du type de ressource utilisé.

Pour les ressources exploitées à l'aide d'un gestionnaire de traitement par lots, le superviseur de tâches récupère les informations d'exécution grâce au gestionnaire de traitement par lots. Les informations récupérées dépendent bien entendu du gestionnaire de traitement par lots utilisé, mais d'une manière générale elles se résument à un état qui détermine si l'exécution est en cours ou non, ceci en surveillant l'exécution du premier processus système de la tâche.

Pour les ressources exploitées en mode interactif, le superviseur de tâche que nous avons mis en œuvre peut avoir deux comportements. Avec son comportement simple, il surveille les

tâches de la même façon qu'un gestionnaire de traitement par lots, c'est-à-dire qu'il ne surveille que le premier processus de chaque tâche. Avec son comportement avancé, il surveille tous les processus d'une tâche.

Comportement simple Lorsqu'une tâche est exécutée (avec les fonctions `fork()` et `exec()`), le superviseur de tâche enregistre le `PID` du premier processus système exécuté. Ensuite la fonction `waitpid()` est exécutée pour attendre la fin de l'exécution du processus dont l'identifiant est PID. En fonction du code de retour de `waitpid(PID)`, le superviseur de tâche retourne au gestionnaire d'application qu'il y a eu une défaillance ou que l'exécution s'est convenablement déroulée.

Comportement avancé En collaboration avec Thomas Ropars durant son stage de Master de recherche [148], nous avons proposé un mécanisme de surveillance avancé des tâches. Ce mécanisme repose sur la surveillance fine de chaque processus système qui constitue une tâche. Cela permet de détecter une terminaison anormale même si ce n'est pas le processus père qui connaît une défaillance et cela permet également de déterminer avec précision l'origine de la défaillance.

Pour mettre en œuvre le comportement avancé, nous avons implémenté une fonctionnalité permettant d'intercepter en cours d'exécution les appels aux fonctions liées au cycle de vie d'un processus système : `fork()`, `exit()`, `wait()` et `waitpid()`. Cette interception a été réalisée en utilisant une librairie dynamique contenant une version modifiée de ces quatre fonctions. Cette librairie dynamique est chargée à l'exécution de la tâche en positionnant la variable d'environnement `LD_PRELOAD` et sa portée permet de sur-définir les quatre fonctions pour tous les processus systèmes créés par une tâche. À tout instant, le superviseur de tâches connaît les processus système en cours d'exécution, et cela de façon passive puisqu'il n'a pas besoin de scruter le répertoire `/proc` pour s'apercevoir de la création de processus. La détection d'une défaillance des processus système créés après l'exécution de la tâche est donc possible.

Cette méthode permet de surveiller très finement l'exécution d'une tâche mais possède toutefois une limitation puisqu'elle n'est pas utilisable avec des binaires compilés statiquement. Dans ce cas, le pré-chargement d'une libraire dynamique sur-définissant les fonctions liées au cycle de vie d'un processus n'a aucun effet puisque le code des librairies, en l'occurrence celui de la `libc`, est directement inclus dans l'exécutable. Une solution pourrait être de mettre cela en œuvre directement dans le noyau du système d'exploitation en sur-définissant les fonctions `do_fork()`, `do_exit()` et `do_wait()`. L'inconvénient d'une telle solution réside dans son niveau d'intrusion qui n'est pas envisageable si le système d'exploitation des nœuds ne peut pas être modifié.

6.2 Évaluation

Dans cette partie, nous présentons un ensemble d'évaluations permettant de valider les propositions que nous avons présentées dans cette thèse.

	Bordeaux	Grenoble	Lille	Lyon	Nancy	Orsay	Rennes	Sophia	Toulouse
Bordeaux	-	17,1	10,8	18,4	12,6	8,3	8	10,3	3,8
Grenoble	17,1	-	12,8	3,2	13,2	14,9	15,2	9,8	10,7
Lille	10,8	12,8	-	10,2	9,2	4,3	11,2	16,8	17,4
Lyon	18,4	3,2	10,2	-	10,5	9,1	12,5	7,2	7,9
Nancy	12,6	13,2	9,2	10,5	-	5,6	11,6	17,1	17,8
Orsay	8,3	14,9	4,3	9,1	5,6	-	8,8	28,9	15,5
Rennes	8	15,2	11,2	12,5	11,6	8,8	-	19,1	19,8
Sophia	10,3	9,8	16,8	7,2	17,1	28,9	19,1	-	14,5
Toulouse	3,8	10,7	17,4	7,9	17,8	15,5	19,8	14,5	-

TAB. 6.2 – Matrice du RTT en ms entre les sites de Grid'5000

Ces évaluations ont été réalisées grâce au prototype que nous avons mis en œuvre sous deux formes : en conditions d'exécution réelle et en simulation.

Nous décrivons tout d'abord la plate-forme expérimentale sur laquelle nous avons conduit les évaluations en exécution réelle. Nous présentons ensuite des évaluations qui portent sur le passage à l'échelle des fondations pair-à-pair du système Vigne, les propriétés du protocole de découverte de ressources optimisé pour l'allocation des ressources, l'efficacité de l'allocation et de la co-allocation des ressources et l'exécution fiable d'applications.

6.2.1 Plate-forme pour les exécutions réelles et protocole expérimental

Afin de conduire nos évaluations en exécution réelle, nous avons utilisé la plate-forme Grid'5000 [9, 57] présentée dans la partie 2.1.2. La plupart des nœuds de la grille sont dotés de bi-processeurs simple ou double cœurs d'architecture x86-64 ou EMT64 cadencés de 2 à 2.4 Ghz et de 1 à 4 Go de mémoire vive. Le tableau 6.2 présente les valeurs de temps d'aller/retour d'un paquet réseau de 64 octets (en anglais *Round-Trip Time* – RTT) entre chaque site relevées le 20 juin 2007.

L'utilisation des ressources de la plate-forme Grid'5000 nécessite l'utilisation de l'outil OAR [62] qui permet la réservation de ressources et l'exécution interactive ou différée de tâches sur un site. Une extension d'OAR nommée OAR Grid permet de réserver des ressources sur tous les sites de la grille.

Chaque site de Grid'5000 gère de façon autonome l'ensemble des ressources qu'il possède. Ainsi, chaque site possède un système de fichier distribué indépendant pour les données des utilisateurs. De plus chaque site est libre d'installer le système d'exploitation qu'il souhaite avec la seule contrainte qu'il soit fondé sur Linux. Cette dernière caractéristique est problématique pour nos évaluations car les systèmes étant hétérogènes, des versions différentes des librairies peuvent être installées sur les ressources. Cela implique une compilation spécifique du prototype et des applications que nous exécutons pour chaque version de système. Toutefois, nous avons utilisé l'outil Kadeploy [57] proposé par la plate-forme pour déployer un nouveau système d'exploitation sur les ressources et ainsi avoir un système d'exploitation homogène sur l'ensemble des ressources.

Le protocole expérimental pour les évaluations en exécution réelle est constitué des étapes suivantes :

1. réservation des ressources avec OAR ou OAR Grid ;
2. déploiement avec Kadepoy d'un environnement personnalisé contenant un système d'exploitation fondé sur Linux et Vigne ;
3. démarrage de Vigne sur tous les nœuds ;
4. exécution d'un script permettant de soumettre des tâches et de changer éventuellement des paramètres de Vigne à chaud ;
5. récupération des fichiers journaux ;
6. libération des ressources.

Pour ces expériences, nous avons utilisé le système d'exploitation GNU/Linux Debian Etch doté d'un noyau Linux en version 2.6.21.

Lors d'une expérience, chaque nœud produit un fichier journal qui contient les informations relatives à l'exécution des applications et la bande passante utilisée pour chaque service. À la fin d'une expérience, tous les fichiers journaux sont collectés et analysés à l'aide de scripts que nous avons développés avec les langages bash et awk.

6.2.2 Passage à l'échelle de l'infrastructure

Dans les chapitres 3 et 4, nous avons vu que le système Vigne est fondé sur des réseaux logiques structurés et non structurés. Ces fondations confèrent au système des propriétés d'auto-organisation et d'auto-réparation dans un environnement où les ressources sont volatiles. Pour garantir ces propriétés en dépit de la volatilité, des opérations de maintenance sont continuellement exécutée par le système. L'objectif est de maintenir à jour la vue locale que chaque nœud possède de la grille. Le nombre d'opérations de maintenance augmente avec l'augmentation de la volatilité des ressources.

En collaboration avec Louis Rilling, nous avons évalué le coût de cette maintenance dans un environnement volatile où la durée de vie des nœuds suit une loi exponentielle et où les apparitions des nœuds suivent une loi de Poisson. Compte-tenu des paramètres que nous avons choisis, un nœud apparaît en moyenne toutes les 35 minutes et reste en moyenne 30 minutes connecté.

Nous avons mesuré la bande passante consommée pour la maintenance des réseaux logiques dans une grille composée de 500, 1000 et 2000 nœuds. Pour cette évaluation nous avons émulé des nœuds Vigne sur 100 nœuds physiques du site rennais de Grid'5000 (64 nœuds de la grappe paraci et 36 de la grappe parasol). L'émulation est une exécution en conditions réelles mais plusieurs nœuds Vigne sont placés sur un nœud physique. Sachant que nous mesurons des bandes passantes au niveau des réseaux logiques, ce type d'exécution n'a aucune incidence sur les mesures et permet de simplifier le déroulement de l'expérience puisque moins de nœuds physiques sont utilisés.

La figure 6.6 présente le nombre de nœuds connectés à un instant donné en fonction du nombre maximal de nœuds dans le système. La figure 6.7 présente la bande passante totale consommée à un instant donné pour les opérations de maintenance des réseaux logiques. La

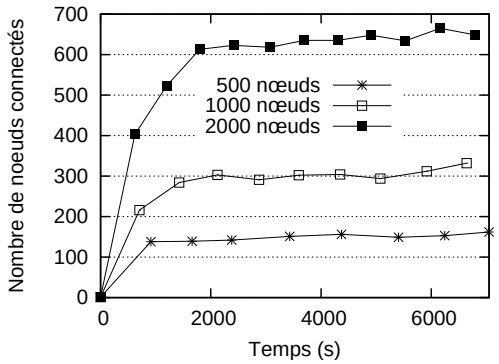


FIG. 6.6 – Évaluation de l'infrastructure P2P : nombre de nœuds connectés

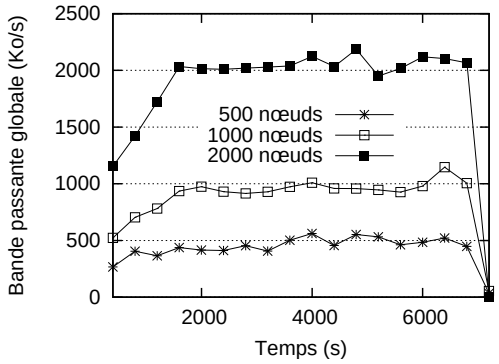


FIG. 6.7 – Évaluation de l'infrastructure P2P : bande passante globale consommée

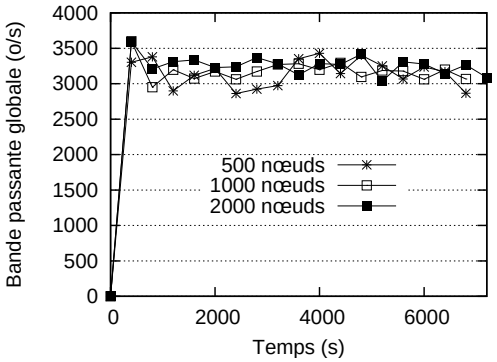


FIG. 6.8 – Évaluation de l'infrastructure P2P : bande passante consommée en moyenne par nœud

figure 6.8 présente le ratio entre la bande passante totale consommée et le nombre de nœuds connectés, ce qui donne une bande passante moyenne par nœud.

Ces résultats montrent que la bande passante moyenne par nœud reste constante lorsque le nombre de nœuds dans le système augmente et que cette valeur reste faible, de l'ordre 3,3 Ko/s.

De plus, il faut noter que les valeurs que nous avons choisies pour la volatilité des nœuds sont très pessimistes. En effet, dans une grille de production, la volatilité est bien plus faible. À titre uniquement indicatif, nous avons étudié la base de données OAR des défaillances des nœuds de la grappe d'expérimentation parasol du site rennais de Grid'5000 qui se sont produites dans l'année 2006. Nous avons mesuré qu'en moyenne, chaque nœud subit 46,5 défaillances (une défaillance tous les 7,8 jours) et chaque défaillance dure 9h24.

Les valeurs que nous avons observées permettent de majorer le trafic généré par les opérations de maintenance et confirment la capacité de passage à l'échelle des fondations pair-à-pair que nous avons utilisées.

6.2.3 Comparaison de différents protocoles de découverte de ressources

Dans cette partie nous avons étudié trois protocoles de découverte de ressources qui sont : (1) un protocole de marches aléatoires, (2) un protocole de marches aléatoires enrichi d'un cache ayant une politique de gestion de type *apprentissage* comme celui proposé dans [99] et (3) le protocole optimisé que nous avons proposé dans la partie 4.2.2. Les deux protocoles utilisés pour la comparaison avec notre proposition ont été choisis pour les raisons suivantes. Le protocole 1 est un protocole basique permettant de limiter de façon significative le coût, du point de vue de la bande passante réseau consommée, d'une découverte de ressources par rapport à l'inondation de requêtes puisque la recherche peut être arrêtée dès que suffisamment de réponses sont reçues. Ce protocole n'est doté d'aucune intelligence et le résultat d'une découverte de ressources est purement statistique. Le protocole 2 est une optimisation du protocole 1 qui inclut une forme d'intelligence en utilisant des caches dotés d'une politique de gestion spécifique. Dans la famille des protocoles utilisant des marches aléatoires, ce protocole est reconnu par la communauté scientifique et il est cité dans de nombreux travaux. Finalement, les protocoles 1 et 2 sont adaptés à la découverte de ressources fondée sur des critères multiples qui peuvent être définis sur une plage de valeurs et dont le nombre peut être variable.

Ces protocoles de découverte de ressources sont paramétrés par différentes valeurs comme : la profondeur d'une marche aléatoire, la durée maximale d'une découverte de ressources, le nombre de résultats demandés pour une découverte de ressources, la taille des caches ou encore la durée de vie d'une information dans un cache. Pour les trois protocoles nous avons fixé la profondeur d'une marche aléatoire à 6, la durée maximale d'une découverte de ressources à 60 UTV. Pour les protocoles 2 et 3, nous avons fixé la taille des caches à 200 entrées et la durée de vie des éléments dans le cache à une heure. Dans toutes les expériences, nous avons fait varier le nombre de résultats demandés à la suite d'une découverte de ressources entre 1 et 6.

Les protocoles de découverte de ressources ont des comportements dépendants de leur contexte d'utilisation. Ainsi, d'autres paramètres interviennent dans nos expériences : le nombre de nœuds dans le système, la capacité de calcul d'un nœud, le nombre de tâches soumises, le nombre d'instructions d'une tâche, l'intervalle de soumission entre les tâches et la rareté des ressources à trouver. Nous avons fixé la capacité de calcul d'un nœud à une instruc-

tion par UTV et l'intervalle de soumission des tâches à 5 UTV. Les autres paramètres pouvant être fixés ou variables selon l'expérience.

Compte-tenu du grand nombre de paramètres à faire varier, beaucoup d'expériences ont dû être réalisées. Nous avons choisi d'utiliser le mode de simulation de Vigne afin de réduire le temps d'expérimentation et de pouvoir effectuer des comparaisons avec une topologie strictement identique. Il est en effet délicat de mobiliser pendant une période supérieure à 12 heures un grand nombre de nœuds sur Grid'5000 compte-tenu du nombre d'utilisateurs simultanés. De plus, il est encore plus compliqué d'obtenir des nœuds identiques entre deux réservations de ressources pour effectuer des comparaisons équitables.

Nous avons évalué plus particulièrement l'incidence de la charge des ressources, l'incidence de la diversification du type des ressources et le temps de découverte de ressources.

6.2.3.1 Incidence de la charge des ressources

Nous avons étudié le comportement des protocoles de découverte de ressources vis-à-vis d'une charge de travail variable.

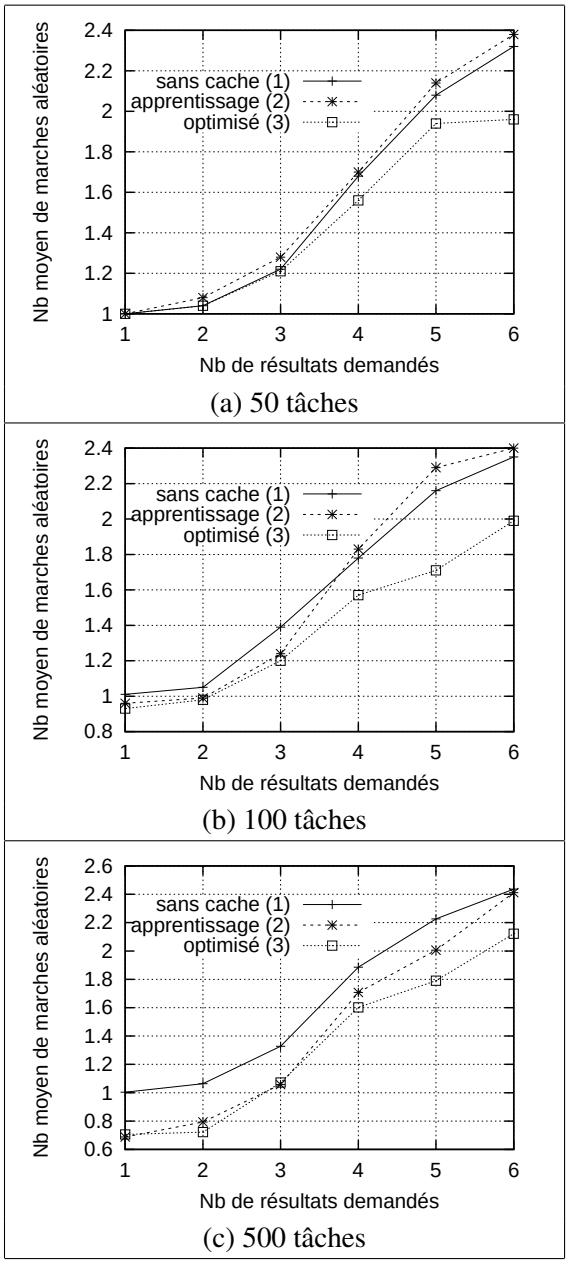
Nous avons réalisé 108 simulations où nous avons fait varier le nombre de tâches soumises au système (50, 100 et 500), la charge de travail d'une tâche (100 ou 500 instructions), le nombre de résultats demandés à la suite d'une découverte de ressources (de 1 à 6) et le protocole de découverte de ressources (protocole de marches aléatoires sans cache (protocole 1), protocole de marches aléatoires avec cache (protocole 2) et stratégie d'apprentissage et protocole optimisé (protocole 3)).

Le nombre de nœuds dans le système a été fixé à 500 pour toutes les simulations.

Dans la première série de simulations (cf. figure 6.9), nous avons fixé la charge de travail d'une tâche à 100 instructions et dans la seconde série (cf. figure 6.10), nous avons fixé la charge de travail d'une tâche à 500 instructions. Nous avons mesuré le nombre moyen de marches aléatoires nécessaires à la découverte des ressources pour chaque simulation, ce qui nous donne une bonne idée de la bande passante consommée par chacun des protocoles.

D'une façon générale, nous pouvons constater que lorsque que le nombre de tâches soumises augmente et que lorsque le nombre de résultats demandés pour une découverte de ressources augmente, le nombre moyen de marches aléatoires augmente, même si la profondeur d'une marche aléatoire est fixée à 6. Cela est dû au fait que lorsque des ressources sont utilisées par l'exécution de tâches, elles ne peuvent plus répondre aux requêtes de découverte de ressources et donc plus d'une marche aléatoire peut être nécessaire pour trouver les ressources requises.

Lorsque les ressources ne sont pas très chargées (cf. figure 6.9.a), le même nombre de marches aléatoires est effectué pour les trois protocoles si le nombre de résultats demandés est inférieur à 4. À partir de 4 résultats demandés, le protocole 3 est plus économe en marches aléatoires que les 2 autres et pour 6 résultats demandés, il permet d'économiser près de 19% de marches aléatoires. Ce résultat s'explique par le fait que les caches utilisés par le protocole 3 ne sont pas suffisamment remplis lorsque peu de résultats sont demandés. Concernant le protocole 2, les caches ne sont d'aucune utilité puisque le nombre d'informations enregistrées est trop faible compte-tenu du fait que ce protocole ne bénéficie pas de la fonctionnalité de dissémination d'information comme nous l'avons proposé dans notre protocole optimisé. Nous pouvons

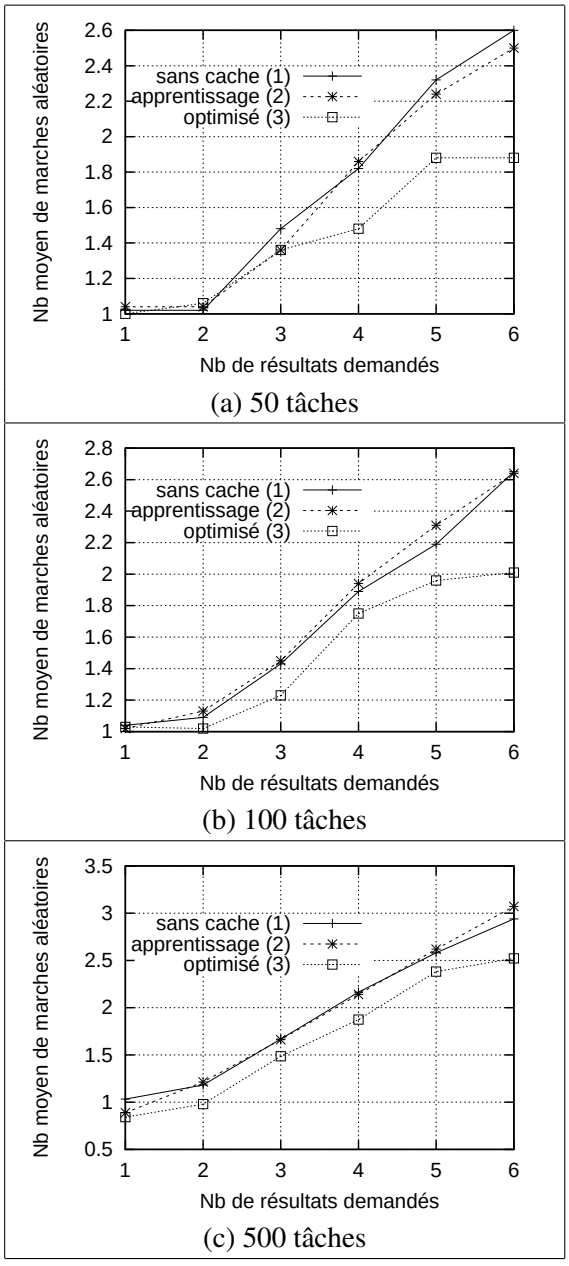


(a) 50 tâches

(b) 100 tâches

(c) 500 tâches

FIG. 6.9 – Incidence de la charge des ressources pour une grille composée de 500 nœuds et de tâches comportant 100 instructions



(a) 50 tâches

(b) 100 tâches

(c) 500 tâches

FIG. 6.10 – Incidence de la charge des ressources pour une grille composée de 500 nœuds et de tâches comportant 500 instructions

également noter que le protocole 2 peut s'avérer plus coûteux que le protocole 1. Cela est dû à une optimisation que nous avons réalisée sur les protocoles utilisant un cache, les protocoles 2 et 3, pour augmenter la vitesse de découverte de ressources, au détriment d'un léger surcoût de bande-passante. Ce phénomène se produit lorsque le nœud qui initie une découverte possède un nombre de ressources dans son cache qui est inférieur au nombre de résultats demandés et que l'interrogation de ressources en cache s'avère infructueuse. Dans ce cas, une marche aléatoire peut être initiée pour rien, mais permet de gagner le temps correspondant au délai accordé pour attendre les réponses d'une marche aléatoire que nous avons fixé à 5 UTV.

Pour une exécution de 100 tâches (cf. figure 6.9.b), le protocole 2 permet d'économiser de la bande passante pour un nombre de résultats demandés compris entre 1 et 3. Au delà, les caches ne sont pas suffisamment remplis pour obtenir 4 résultats et plus, et ce protocole souffre du problème énoncé dans le paragraphe précédent. Quant au protocole 3, il est le plus économe dans tous les cas, et permet une économie significative de bande passante à partir de 4 résultats demandés.

Pour une exécution de 500 tâches (cf. figure 6.9.c), les protocoles 2 et 3 ont une efficacité similaire pour un nombre de résultats demandés inférieur à 4. Le bon résultat du protocole 2 est dû au grand nombre de tâches exécutées qui permet aux caches d'être suffisamment remplis. Pour un nombre de résultats demandés supérieur à 3, l'efficacité du protocole 2 diminue par rapport au protocole 3. Nous pouvons noter que les deux protocoles utilisant un cache ne requièrent parfois aucune marche aléatoire, ce qui explique dans la figure 6.9.c les valeurs du nombre moyen de marches aléatoires entre 0,7 et 0,8 lorsque le nombre de résultats demandés est inférieur à 3.

Des remarques similaires s'appliquent pour le cas où les tâches ont une charge de 500 instructions, toutefois le protocole 2 donne de moins bons résultats. Cela est dû au fait que les données enregistrées dans les caches ne sont pas réellement utiles puisque les ressources allouées restent occupées 5 fois plus longtemps que dans la première série de simulation. En effet, l'interrogation des ressources connues dans les caches conduit souvent à un échec puisque elles sont encore occupées.

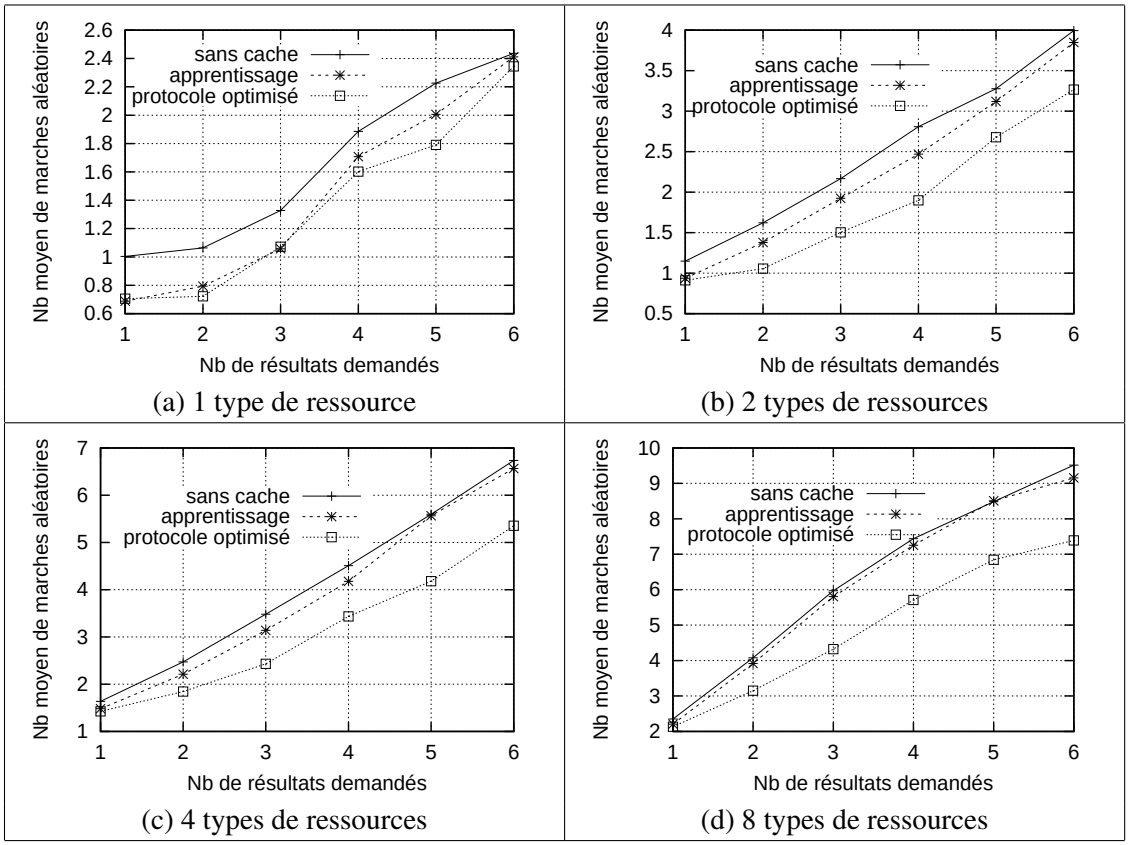
Dans ces simulations, nous pouvons observer que le protocole optimisé que nous avons proposé obtient systématiquement des résultats égaux ou meilleurs que les autres protocoles pour ce qui concerne la bande passante consommée. Dans certains cas, le gain peut atteindre 27% (cf. figure 6.10.a).

D'une manière générale, notre protocole offre les meilleurs résultats lorsque les conditions de recherche de ressources libres sont les plus complexes, c'est-à-dire lorsqu'il n'y a plus beaucoup de ressources disponibles.

6.2.3.2 Incidence de la rareté des ressources

Nous avons étudié le comportement des protocoles de découverte de ressources vis-à-vis de la rareté des ressources à découvrir.

Nous avons effectué 72 simulations où nous avons exécuté 500 tâches comportant chacune 100 instructions sur une grille de 500 nœuds. Nous avons fait varier le nombre de résultats demandés, le protocole utilisé et la rareté des ressources qui composent la grille. Les ressources sont classées en n types avec $n \in \{1,2,4,8\}$ et il y a $\frac{500}{n}$ ressources de chaque type. Chaque



(a) 1 type de ressource (b) 2 types de ressources

(c) 4 types de ressources (d) 8 types de ressources

FIG. 6.11 – Étude de l'incidence de la rareté des ressources pour une grille composée de 500 nœuds et pour l'exécution de 500 de 100 instructions

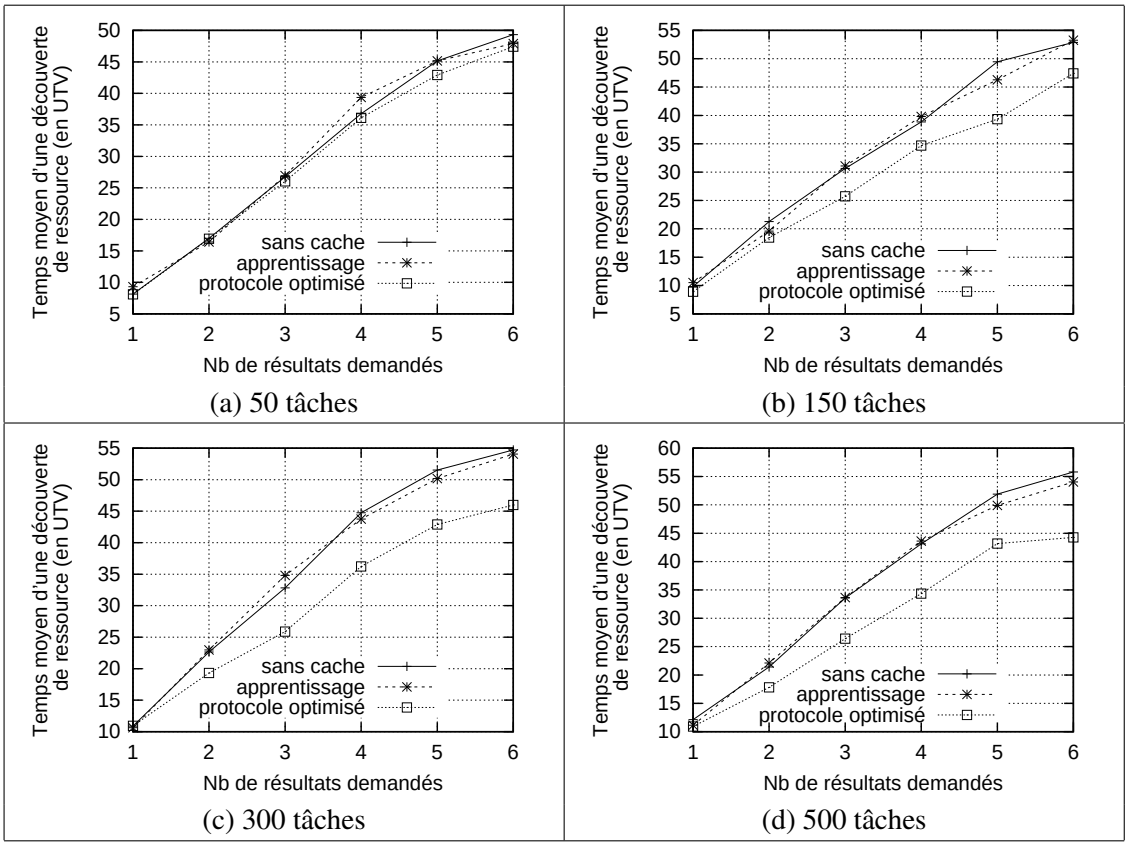


FIG. 6.12 – Temps de découverte des ressources pour une grille composée de 500 nœuds et pour l'exécution de tâches durant 500 UTV

tâche exécutée nécessite une ressource d'un des n types sachant que nous avons affecté $\frac{500}{n}$ tâches pour chaque type.

La figure 6.11 présente les résultats de ces simulations. Nous pouvons voir que le nombre moyen de marches aléatoires nécessaires augmente avec la rareté des ressources pour les trois protocoles. Le protocole 2 perd en efficacité comparativement au protocole 3 lorsque les ressources deviennent de plus en plus rares. Ainsi, lorsque le système est composé de 4 types ressources ou plus, l'efficacité du protocole 2 est presque identique à celle du protocole 1.

À partir de 2 résultats demandés, le protocole optimisé que nous avons proposé est systématiquement plus efficace que les deux autres, et cela même lorsque la rareté des ressources augmente. Cela est du au fait que l'utilisation du protocole optimisé que nous proposons permet d'augmenter de façon significative le contenu des caches en termes d'informations liées à des nœuds non alloués. Par conséquent, de nombreuses requêtes de découverte de ressources par marches aléatoires peuvent être évitées.

6.2.3.3 Temps de découverte des ressources

Nous avons également mesuré le temps des découvertes de ressources. Pour cela nous avons réalisé 72 simulations où nous avons exécuté un nombre variable de tâches (50, 150, 300 et 500) comportant 500 instructions sur une grille de 500 nœuds. De plus, nous avons divisé les ressources de la grille en 8 types afin d'augmenter la complexité de la recherche de ressources.

La figure 6.12 montre les résultats de ces simulations. Nous pouvons voir que pour 50 tâches soumises (cf. figuree 6.12.a), les protocoles nécessitent en moyenne le même temps pour réaliser une découverte de ressources. Pour un nombre plus élevé de tâches, le protocole 3 se démarque des deux autres et permet de gagner de 4 à 12 UTV en moyenne pour les découvertes de ressources d'une simulation, ce qui correspond à un gain de temps pouvant aller jusqu'à 28%.

6.2.3.4 Conclusion sur l'évaluation du protocole de ressources optimisé

Lors de ces simulations, nous avons comparé le protocole de découverte de ressources présenté dans la partie 4.2.2 de cette thèse au protocole basique de marches aléatoires et à un protocole de l'état de l'art qui est fondé sur l'utilisation des marches aléatoires et de caches gérés par une stratégie d'apprentissage.

En ayant imposé que la découverte de ressources permette de trouver des ressources libres, nous n'avons pas évalué l'efficacité de l'allocation ultérieure puisque cette dernière s'effectue systématiquement de façon à ce qu'au plus une tâche soit placée par processeur à un instant donné. Nous avons donc mesuré le nombre de marches aléatoires nécessaires et le temps de découverte de ressources requis pour que l'allocation de ressources soit optimale.

Dans la plupart des cas, le protocole que nous avons proposé est plus performant que les deux autres, c'est-à-dire qu'il nécessite moins de marches aléatoires et qu'il permet de découvrir les ressources libres plus rapidement. Ces propriétés restent vraies lorsque la rareté des ressources augmente, soit à cause de la charge globale de la grille qui augmente, soit à cause du type de ressource recherché.

Ces évaluations nous permettent de penser que le protocole que nous avons proposé est un constituant tout à fait adapté pour établir un système d'information de grille complètement distribué.

6.2.4 Efficacité de l'allocation de ressources dans Vigne

Pour exécuter le plus rapidement possible un ensemble de tâches, il faut répartir ces tâches sur l'ensemble des ressources disponibles afin de limiter les exécutions concurrentes sur une même ressource. Un ordonnanceur efficace sera capable de tirer profit de l'ensemble des ressources en répartissant au mieux les tâches sur la grille.

Nous avons mesuré l'efficacité de l'allocation de ressources dans Vigne. Cette évaluation nous permet de valider les mécanismes d'allocation de ressources mais également le système d'information sur lequel l'allocation de ressources repose.

Nous avons réalisé cette évaluation en conditions réelles d'exécution. Pour cela nous avons déployé Vigne sur 449 nœuds bi-processeurs répartis sur 5 sites de la plate-forme Grid'5000. Le tableau 6.3 présente la répartition des ressources sur les différentes grappes utilisées.

Site	Grappe	Nombre de nœuds	Nombre de CPU utilisés	Temps d'exécution de l'application de test (s)
Nancy	grelon	59	118	422
	grillon	28	56	342
Orsay	gdx	103	206	430
Lyon	sagittaire	46	92	348
Rennes	paraquad	38	76	289
	parasol	2	4	377
	paravent	88	176	341
Sophia	azur	51	102	431
	helios	13	26	312
	sol	21	42	262
Total		449	898	–

TAB. 6.3 – Répartition des nœuds utilisés pour l'évaluation de l'allocateur de ressources de Vigne

Nous avons exécuté un ensemble de tâches séquentielles identiques sur les ressources et nous avons mesuré le temps d'exécution de chaque tâche. Le tableau 6.3 présente également le temps d'exécution d'une tâche sur chaque grappe utilisée. L'objectif de cette évaluation est de s'assurer que le service d'allocation de ressources de Vigne profite de l'ensemble des ressources disponibles pour exécuter les tâches en ne plaçant plusieurs tâches par processeur que lorsqu'aucun processeur n'est libre.

Nous avons effectué trois expériences où nous avons progressivement augmenté la charge de travail sur la grille. Tout d'abord nous avons lancé 898 tâches à raison d'une tâche par seconde, ensuite nous avons lancé 1796 tâches à raison de 2 tâches par seconde et finalement nous avons lancé 3592 tâches à raison de 4 tâches par seconde. Pour ces expériences nous avons utilisé le protocole de marches aléatoires optimisé présenté dans la partie 4.2.2. La profondeur des marches aléatoires a été fixée à 8, le nombre de résultats demandés pour une découverte de ressources a été fixé à 32 et le temps d'une découverte de ressources a été borné à 60 secondes.

Les figures 6.13, 6.14 et 6.15 montrent les temps d'exécution des tâches pour chaque expérience. Nous précisons que dans ces résultats, les numéros de tâches ne sont pas ordonnés dans le temps.

Lorsque 898 tâches sont soumises, c'est-à-dire lorsque qu'il y a autant de tâches soumises que de processeurs, d'après la figure 6.13 que la répartition des temps d'exécution des tâches suit la répartition des ressources utilisées. Pour aider la lecture de ces résultats, nous donnons un histogramme de la répartition des ressources pour le temps d'exécution de l'application de test dans la figure 6.16. De plus, la figure 6.17 présente la répartition des temps d'exécution pour le sac de 898 tâches sous la même forme. Nous pouvons voir plus simplement que la répartition est presque identique. Aucun processeur n'a reçu plus d'une tâche à exécuter à la fois. Mais les processeurs les plus rapides ont pu exécuter plusieurs tâches lors de l'expérience. Nous pouvons remarquer que les temps obtenus sont légèrement différents des temps de référence car nous avons mesuré le temps total depuis la soumission des tâches, ce qui inclut la phase de découverte de ressources. Dans cette expérience, les ressources ne sont pas saturées puisqu'il y a un intervalle d'une seconde entre la soumission des tâches. Ainsi, lorsque la seconde moitié

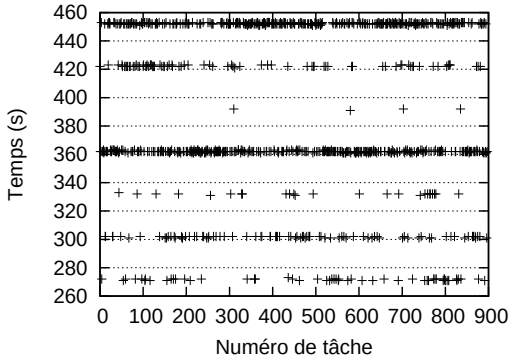


FIG. 6.13 – Temps d'exécution des 898 tâches, soumission d'une tâche par seconde

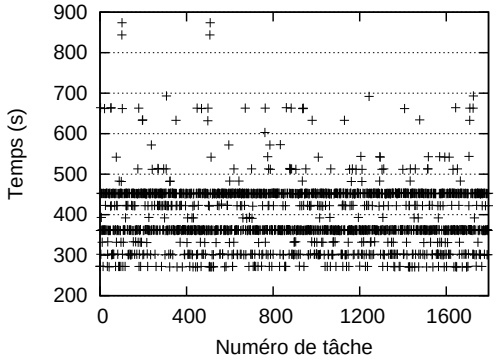


FIG. 6.14 – Temps d'exécution des 1796 tâches, soumission de deux tâches par seconde

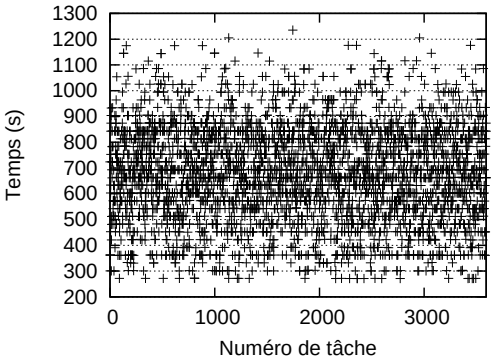


FIG. 6.15 – Temps d'exécution des 3592 tâches, soumission de quatre tâches par seconde

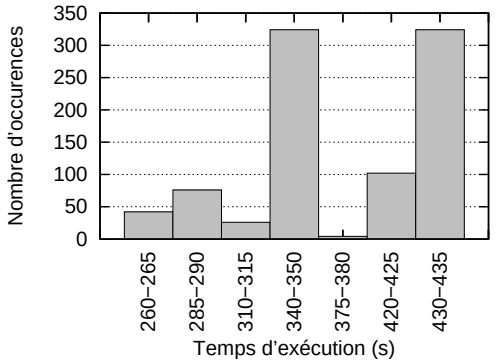


FIG. 6.16 – Répartition des nœuds de calculs utilisés en fonction de leur vitesse d'exécution de l'application de test

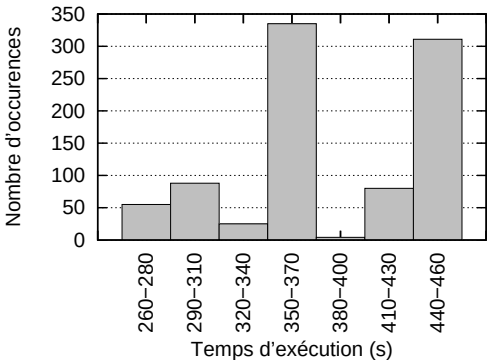


FIG. 6.17 – Répartition des temps d'exécution des 898 tâches

des tâches commence à être soumise, les premières tâches terminent leur exécution et libèrent des ressources.

Lorsque 1796 tâches sont soumises, nous pouvons voir sur la figure 6.14 que la majorité des exécutions a lieu sur des processeurs n'exécutant qu'une seule tâche à la fois (puisque la répartition des temps d'exécution suit globalement la répartition des ressources) et qu'une minorité d'exécutions (environ 4,5%) a lieu sur des processeurs exécutant 2 tâches en concurrence. Avec un ordonnanceur ayant une connaissance globale des ressources, les exécutions pourraient toutes se dérouler sur des processeurs n'exécutant qu'une tâche à la fois. Avec Vigne, lorsque les ressources deviennent très chargées, c'est-à-dire à partir de la moitié de l'expérience, les ressources libres sont plus difficiles à trouver. Afin de limiter la bande passante consommée pour la découverte des ressources, nous avons limité à 32 le nombre de résultats demandés suite à une découverte de ressources, ce qui n'a pas permis dans certains cas d'aboutir à une allocation sur un processeur libre à cause du service au mieux proposé par l'ordonnanceur distribué de Vigne.

Lorsque 3592 tâches sont soumises, nous pouvons voir sur la figure 6.15 que la majorité des tâches s'exécute sur des processeurs exécutant deux tâches simultanément puisque les temps d'exécution sont globalement doublés par rapport aux temps de référence. Ici les ressources sont complètement saturées. Un faible nombre de tâches a été exécuté sur des processeurs exécutant trois tâches simultanément. Cela est, comme dans le cas précédent, imputable au nombre de résultats demandés que nous avons fixé pour une découverte de ressources.

Ces expériences permettent de montrer que l'ordonnancement complètement décentralisé du système Vigne permet d'obtenir une allocation de ressources qui génère un placement des tâches permettant presque d'occuper l'intégralité des ressources. Ainsi, la répartition de la charge est proche d'une répartition parfaite qui pourrait être obtenue avec un ordonnanceur centralisé ayant la connaissance de toutes les ressources, même lorsque les ressources se retrouvent saturées. De plus, les résultats obtenus pourraient être améliorés en augmentant le nombre de résultats demandés pour une découverte de ressources et la durée d'une découverte de ressources. En effet, l'ajustement de ces paramètres permet d'améliorer l'allocation des ressources en fonction du contexte d'utilisation des ressources d'une grille, le tout étant un compromis entre le temps utilisé pour les découvertes de ressources, la bande passante consommée pour les découvertes de ressources et la qualité des allocations de ressources.

6.2.5 Efficacité de la co-allocation de ressources dans Vigne

Nous avons évalué l'efficacité des mécanismes de co-allocation de ressources du système Vigne. Ces mécanismes présentés dans la partie 4.3.3 permettent d'allouer simultanément plusieurs ressources possédant la meilleure interconnexion réseau possible pour l'exécution de tâches communicantes.

Cette évaluation a été menée en conditions réelles d'exécution sur la plate-forme Grid'5000 où nous avons utilisé 376 nœuds selon la répartition présentée dans le tableau 6.4. Nous avons réalisé 2 types d'expériences où nous avons dans les deux cas exécuté une application utilisant MPI et permettant de calculer le nombre π en parallèle en utilisant ou non les mécanismes de co-allocation spatiale de Vigne. Dans chaque type d'expérience nous avons fait varier le degré de parallélisme de l'application (4, 8, et 16 tâches parallèles par application), nous avons exé-

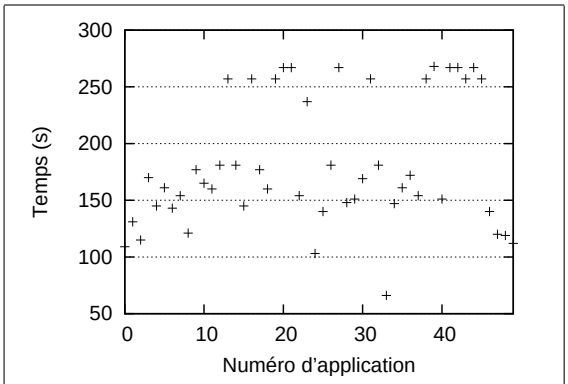
Site	Grappe	Nombre de nœuds	Nombre de CPU utilisés
Lille	–	31	62
Nancy	grillon	42	84
Orsay	gdx	93	186
Lyon	sagittaire	45	90
Rennes	parasol	57	114
	paravent	58	116
Sophia	azur	50	100
Total		376	752

TAB. 6.4 – Répartition des nœuds utilisés pour l'évaluation des mécanismes de co-allocation de ressources de Vigne

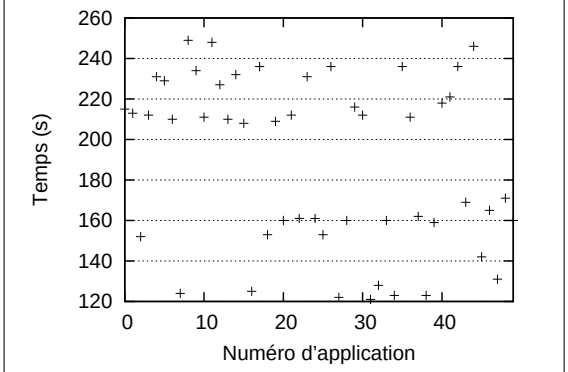
cuté 50 instances de l'application à chaque fois et nous avons mesuré le temps d'exécution de chaque instance d'application. Afin que l'exécution de l'application parallèle soit la plus rapide possible, il faut que toutes les tâches qui la composent soient exécutées sur des ressources proches du point de vue du réseau, c'est-à-dire sur un même site Grid'5000. Dans notre cas, nous n'avons pas utilisé les réseaux intra-grappe rapides de type Myrinet ou Infiniband. Sur le site rennais utilisé dans cette expérience, tous les nœuds de toutes grappes sont sur un unique commutateur pour le réseau IP. Nous pouvons donc supposer que sur le site rennais, les nœuds de grappes différentes possèdent la même connectivité réseau. Les temps de référence (très légèrement variables selon le site Grid'5000 considéré) pour l'exécution de l'application lorsque les tâches sont exécutées sur un même site sont de l'ordre de 65 secondes pour une application composée de 4 tâches, de 45 secondes pour une application composée de 8 tâches et de 34 secondes pour une application composée de 16 tâches.

La figure 6.18 présente les résultats de l'évaluation pour le cas où les mécanismes de co-allocation spatiale n'ont pas été utilisés et la figure 6.19 présente les résultats de l'évaluation lorsque les mécanismes de co-allocation spatiale de Vigne ont été utilisés. Les numéros de tâche d'exécution ne sont pas ordonnés dans le temps.

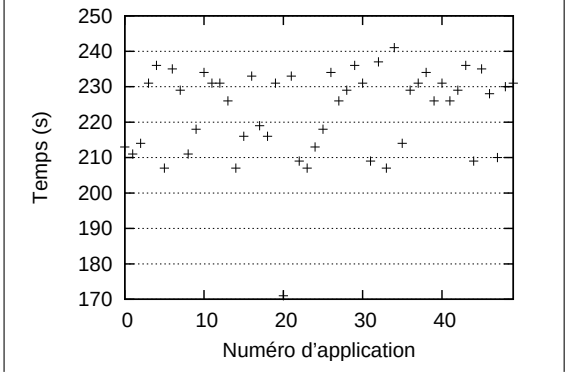
Nous pouvons remarquer que d'une façon générale, les mécanismes de co-allocation spatiale de Vigne permettent de réduire de façon significative le temps d'exécution des applications. Sur les figures 6.19.a et 6.19.b, nous pouvons voir que majoritairement les exécutions se sont déroulées avec le temps d'exécution nominal, c'est-à-dire entre 60 et 68 secondes pour des applications composées de 4 tâches et entre 41 et 46 secondes pour des applications composées de 8 tâches. Par opposition, les figures 6.18.a et 6.18.b montrent un temps d'exécution largement plus grand puisque les tâches ont été distribuées aléatoirement sur tous les sites Grid'5000. Quant à l'exécution d'applications de 16 tâches (cf. figure 6.19.c) la majorité des applications s'exécute de façon optimale lorsque l'on utilise les mécanismes de co-allocation spatiale. Toutefois 17 applications s'exécutent en mode dégradé, c'est-à-dire que les tâches d'une application utilisent plus d'un site Grid'5000. Nous expliquons cela par le fait que lorsque les ressources commencent à être occupées, il est possible qu'il ne reste plus de site où 16 ressources sont libres simultanément. Dans ce cas, nos mécanismes de co-allocation spatiale ne sont plus capables d'allouer des ressources ne provenant que d'un seul site et doivent placer les tâches des applications sur 2 ou 3 sites. Cela dit, comparativement à une exécution qui



(a) 4 tâches par application parallèle

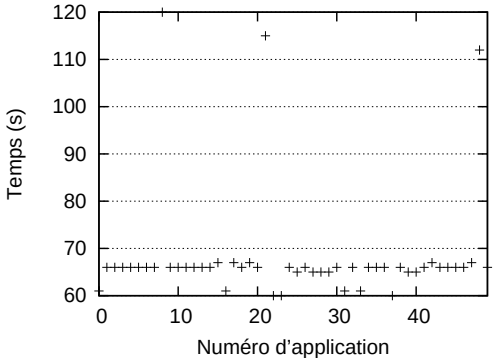


(b) 8 tâches par application parallèle

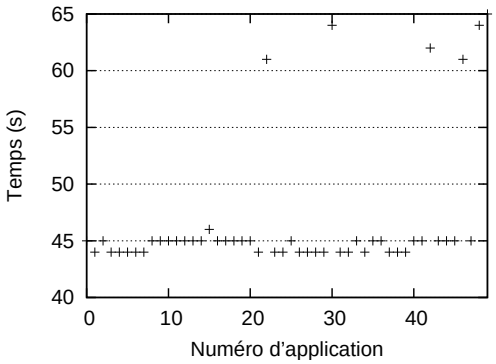


(c) 16 tâches par application parallèle

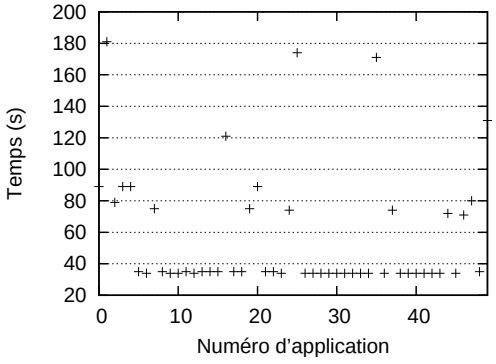
FIG. 6.18 – Évaluation des mécanismes de co-allocation de ressources sans utiliser l'optimisation spatiale



(a) 4 tâches par application parallèle



(b) 8 tâches par application parallèle



(c) 16 tâches par application parallèle

FIG. 6.19 – Évaluation des mécanismes de co-allocation de ressources en utilisant l'optimisation spatiale

	Sans co-allocation sptatiale			Avec co-allocation spatiale		
	4 nœuds	8 nœuds	16 nœuds	4 nœuds	8 nœuds	16 nœuds
1 site	1	0	0	47	45	33
2 sites	14	0	0	3	5	12
3 sites	22	9	0	0	0	2
4 sites	13	24	1	0	0	3
5 sites	0	14	19	0	0	0
6 sites	0	3	30	0	0	0

TAB. 6.5 – Répartition du nombre de sites utilisés pour l'exécution de 50 applications parallèles en fonction du nombre de tâches par application parallèle et en fonction de l'utilisation ou non des mécanismes de Vigne pour la co-allocation spatiale de ressources

n'utilise pas les mécanismes de co-allocation spatiale (cf. figure 6.18.c), les temps d'exécution sont très largement inférieurs. Le tableau 6.5 montre la répartition des exécutions en fonction du nombre de tâches par application parallèle et en fonction de l'utilisation ou non des mécanismes de co-allocation spatiale de ressources. Lorsqu'aucun mécanisme de co-allocation spatiale n'est utilisé, l'ordonnanceur de Vigne affecte les tâches des applications sur n'importe quels nœuds de la grille, ce qui implique que tous les sites peuvent être utilisés simultanément. Avec les mécanismes de co-allocation spatiale, l'ordonnanceur tente systématiquement de placer les tâches d'une application sur des ressources proches, c'est-à-dire les ressources d'un même site, dans la limite bien sûr de l'état de saturation des ressources.

Les résultats obtenus en conditions réelles d'exécution sur la plate-forme Grid'5000 montrent que le système Vigne permet de prendre en compte les contraintes applicatives liées aux communications inter-tâches. Les mécanismes proposés permettent d'optimiser le placement des tâches communicantes afin de garantir l'exécution la plus efficace possible en s'affranchissant si possible d'une forte latence pour la communication entre les nœuds utilisés.

6.2.6 Fiabilité de l'exécution

Nous avons évalué les mécanismes présentés dans la partie 5.2 qui permettent d'exécuter les applications de façon fiable.

Cette évaluation a été menée en conditions réelles d'exécution sur la plate-forme Grid'5000. Nous avons utilisé 402 nœuds de la plate-forme dont la répartition est présentée dans le tableau 6.6.

Nous avons lancé 201 tâches indépendantes à travers Vigne, en espaçant la soumission de chaque tâche d'une seconde. Chaque tâche effectue un calcul synthétique durant 1500 secondes. Une fois que toutes les tâches ont été démarrées, nous avons injecté une série de 100 défaillances sur des nœuds choisis aléatoirement dans la grille. Une défaillance a pour objectif de supprimer les processus applicatifs d'un nœud en leur envoyant le signal `SIGKILL`.

Nous avons mesuré le temps d'exécution de chaque tâche, depuis la soumission jusqu'à la terminaison. La figure 6.20 présente les temps d'exécution des 201 instances de tâches. Dans ces résultats, les numéros de tâche ne sont pas ordonnés dans le temps.

Nous pouvons observer que 152 tâches ont été exécutées dans un temps compris entre 1500 et 1600 secondes. Les autres tâches ont subi des défaillances mais ont été redémarrées. D'après

Site	Grappe	Nombre de nœuds	Nombre de CPU utilisés
Nancy	grelon	39	78
	grillon	19	38
Orsay	gdx	79	158
Rennes	paraquad	50	100
	parasol	50	100
	paravent	89	178
Sophia	azur	66	132
	sol	10	20
Total		402	804

TAB. 6.6 – Répartition des nœuds utilisés pour l'évaluation des mécanismes d'exécution fiable Vigne

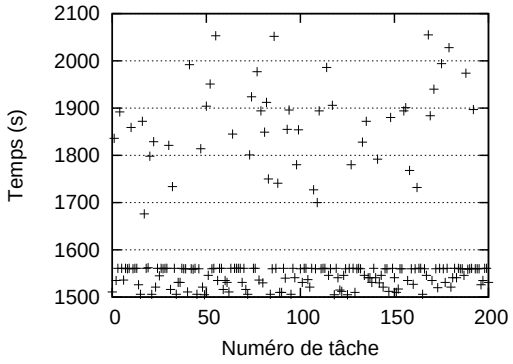


FIG. 6.20 – Répartition des temps d'exécution pour une exécution de 201 tâches avec une injection de 53 défaillances

le système de surveillance de Vigne, 45 tâches ont subi une seule défaillance et 4 tâches ont subi deux défaillances. Au final, toutes les tâches ont été exécutées avec succès.

Ces résultats montrent que Vigne permet de mener à terme l'exécution d'une application en dépit des défaillances liées au contexte d'exécution qui peuvent se produire. Ajouté au mécanisme permettant de détecter la défaillance d'un nœud, ce mécanisme permet de garantir une exécution fiable des applications dans la grille.

6.3 Conclusion

Dans ce chapitre, nous avons présenté des éléments de mise en œuvre des services que nous avons implémentés dans le prototype Vigne. Ce prototype nous a permis d'évaluer, en conditions réelles d'exécution ou par simulation, les contributions présentées dans les chapitres 4 et 5 de cette thèse.

Nous avons tout d'abord évalué la capacité de passage à l'échelle de l'infrastructure pair-à-pair de Vigne. Les évaluations en environnement très volatil permettent de montrer que le coût de maintenance des réseaux pair-à-pair engendré par des reconfigurations perpétuelles et fréquentes n'augmente pas lorsque le nombre de nœuds dans le système augmente et que ce coût est faible compte-tenu de la capacité des liens réseaux d'une grille.

Ensuite, nous avons évalué le système d'information de Vigne. Nous avons comparé le protocole que nous avons proposé dans la partie 4.2.2 à deux autres protocoles de l'état de l'art. Les résultats montrent que notre protocole est plus économe en bande passante et permet de trouver des ressources plus rapidement que les deux autres. En conséquence, ce protocole permet la construction d'un système d'information pour grille auto-organisant, auto-réparant, possédant d'excellentes propriétés de passage à l'échelle et permettant de découvrir avec une forte probabilité des ressources libres, même rares, pour une allocation ultérieure.

Nous avons évalué le service d'allocation de ressources de Vigne. Tout d'abord nous avons mesuré son efficacité pour l'exécution de tâches indépendantes. Les résultats montrent que l'ordonnanceur complètement décentralisé de Vigne permet d'obtenir une répartition des tâches très proche d'un placement parfait qui pourrait être atteint avec un ordonnanceur centralisé. Nous avons ensuite mesuré son efficacité pour l'exécution d'applications composées de tâches communicantes. Les résultats montrent que le système Vigne permet de tenir compte des contraintes liées aux communications entre les tâches d'une application et maximise la probabilité d'allouer des ressources possédant une bonne interconnexion réseau afin de permettre une exécution la plus efficace possible.

Nous avons évalué les mécanismes d'exécution fiable d'applications de Vigne. Ces mécanismes permettent de garantir que les applications exécutées dans Vigne atteignent le terme de leur exécution en dépit des défaillances liées au contexte d'exécution. Pour cela, Vigne permet de surveiller la terminaison anormale de tous les processus système d'une tâche et de redémarrer cette tâche sur une autre ressource lorsqu'une telle terminaison est détectée.

D'une façon générale, ces expérimentations nous ont permis d'évaluer l'ensemble des contributions présentées dans cette thèse, à l'exception de la gestion d'applications composées de tâches ayant des relations de précédence entre-elles. Cependant, cette évaluation est présentée dans le chapitre suivant avec une application industrielle d'EDF R&D.

Chapitre 7

Connexion de la plate-forme de simulation numérique SALOME au système Vigne

Nous présentons dans ce chapitre les travaux que nous avons effectués pour valider notre approche dans un contexte industriel. L'objectif de ces travaux est de réaliser la connexion du système Vigne à la plate-forme de simulation numérique SALOME [31].

Dans un premier temps, nous décrivons la plate-forme SALOME en présentant sa vue d'ensemble et son architecture. Puis, nous proposons une extension à la plate-forme SALOME pour la connexion au système Vigne. Finalement, nous évaluons notre approche avec l'application Saturne/Syrthes développée à EDF R&D qui est une application de type *wokflow* comportant un couplage de codes.

7.1 Description de la plate-forme SALOME

SALOME est une plate-forme logicielle dédiée à la réalisation et au pilotage d'applications de simulation numérique. La plate-forme SALOME est issue d'un co-développement entre plusieurs partenaires dont EDF R&D, CEA et Open CASCADE.

Dans cette partie nous présentons une vue d'ensemble de la plate-forme et nous détaillons son architecture.

7.1.1 Vue d'ensemble de la plate-forme SALOME

La plate-forme SALOME est née de plusieurs besoins industriels concernant la simulation numérique. Tout d'abord, les industriels souhaitaient simplifier la création et le pilotage d'applications modélisant plusieurs physiques comme les applications de couplage de codes. Ensuite, il était nécessaire de simplifier l'intégration de codes métier dans un environnement permettant de les utiliser simplement. Finalement, il fallait définir un standard d'échange de données permettant l'inter-opérabilité des codes de simulation numérique pour en simplifier le couplage.

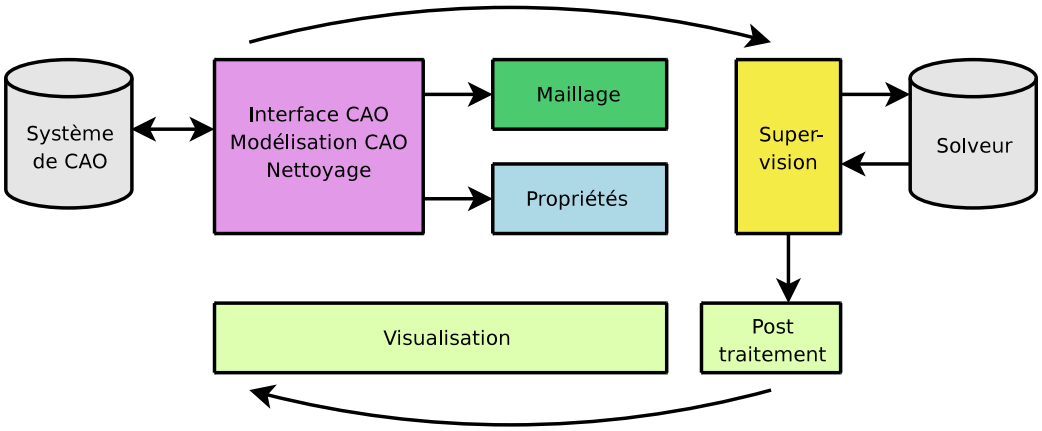


FIG. 7.1 – Interactions entre les fonctionnalités de SALOME

SALOME permet de centraliser l'ensemble des développements d'applications de simulation numérique à EDF R&D ou encore au CEA. D'ores et déjà, la plate-forme est la cible de nombreuses applications métier. Plusieurs projets à EDF R&D ont déjà choisi d'intégrer leurs applications à SALOME. Nous pouvons par exemple citer le projet Code_Aster [5] qui permet de faire des études thermo-mécaniques et plus généralement des modélisations multi-physiques ou encore le projet Code_Saturne [6] qui est un logiciel généraliste de mécanique des fluides.

La figure 7.1 montre les interactions entre les différentes fonctionnalités de SALOME du point de vue de l'utilisateur. Les fonctionnalités offertes sont les suivantes :

- création, modification, import et export d'éléments de *Conception Assistée par Ordinateur* (CAO) ;
- maillage des éléments de CAO dans divers formats et ajout des propriétés physiques aux éléments géométriques ;
- orchestration de calculs faisant intervenir un ou plusieurs solveurs par le superviseur de la plate-forme qui intègre un moteur de *worklflow* permettant de piloter des applications complexes fondées sur une approche composant ;
- visualisation et traitement des résultats.

7.1.2 Architecture

La figure 7.2 présente l'architecture de la plate-forme SALOME. L'architecture peut être découpée en deux niveaux qui sont les services de bas niveau et les modules.

7.1.2.1 Services de bas niveau

Les services de bas niveau de SALOME constituent les fonctionnalités de base de la plate-forme. Ces services sont le noyau, le gestionnaire d'études et l'interface graphique.

Noyau Le noyau fournit les fonctionnalités générales pour gérer les composants SALOME. Par exemple, il permet de créer et de connecter les composants qui peuvent être distribués et

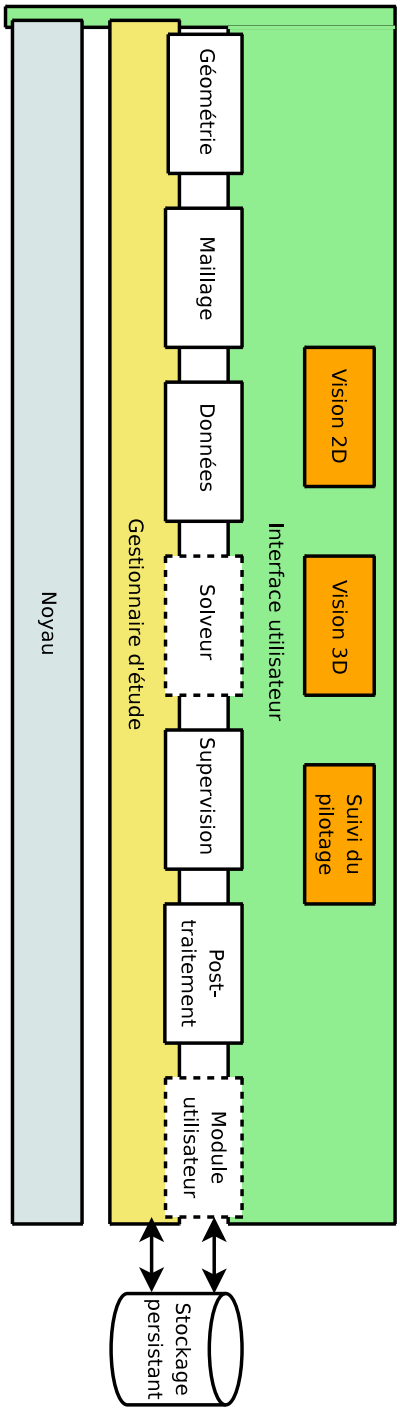


FIG. 7.2 – Architecture de la plate-forme SALOME

il gère la gestion du cycle de vie des applications. En particulier le noyau prend en charge le chargement dynamique de composants distribués, l'exécution de composants ou l'échange de données entre composants. Le noyau de SALOME est fondé sur la technologie CORBA.

Gestionnaire d'étude Le gestionnaire d'étude fournit un environnement générique pour gérer de façon persistante les données partagées par les composants SALOME. Un utilisateur peut créer une étude, ouvrir une étude existante, enregistrer une étude ou encore copier/coller des parties d'une étude.

Interface utilisateur L'interface utilisateur est l'espace de travail de l'utilisateur. Elle fournit les services génériques qui permettent aux modules de SALOME d'intégrer dynamiquement leur propre interface dans le bureau.
Ces services génériques sont par exemple :

- des moteurs de visualisation : un moteur 2D fondé sur la technologie QWT, des moteurs 3D fondés sur les technologies Open CASCADE et VTK,
- des navigateurs d'objets pour les études,
- une console Python qui embarque un interpréteur,
- une console d'affichage,
- un gestionnaire d'enregistrement et restauration de préférences d'affichage,
- un gestionnaire de fenêtre,
- un menu d'aide où chaque module peut ajouter une aide en ligne spécifique.

7.1.2.2 Modules

Les modules sont des composants de haut niveau construits sur les services présentés dans la partie 7.1.2.1. La plate-forme peut être spécialisée par métier. Par exemple, la déclinaison de plate-forme pour le projet Code_Aster [5] comporte les modules génériques de SALOME et des modules spécifiques à Code_Aster. Voyons quelques modules utilisés couramment dans la plate-forme.

Géométrie Le module de géométrie fournit un ensemble de fonctionnalités dédiées à l'édition de modèles CAO. Des primitives permettent de créer des objets géométriques et de réaliser des opérations sur ces objets. Ce module assure la cohérence des modèles pour un maillage ultérieur.

Maillage Ce module a pour but de créer des maillages à partir de modèles créés ou importés par le module de géométrie. Le maillage d'un objet correspond à sa discrétisation afin de pouvoir l'utiliser lors d'une simulation. Dans SALOME, les maillages sont enregistrés au format MED (Module d'Échange de Données).

Données Le module de données permet de paramétrer les propriétés physiques des objets. Ces propriétés ne concernent pas les propriétés de géométrie ou de maillage. Par exemple, le

module de données permet de définir des propriétés comme la conductivité des matériaux, la viscosité ou encore les conditions initiales d'un objet.

Supervision Le module de supervision permet de définir graphiquement et d'exécuter un processus de calcul qui correspond à un graphe acyclique orienté. Le superviseur permet de définir le couplage entre différents composants distribués comme les modules solveurs.

Post-traitement Le module de post-traitement permet à l'utilisateur d'analyser le résultat d'une simulation numérique à l'aide de visualisateurs 2D et 3D. Il permet également d'importer des maillages au format MED pour effectuer des comparaisons entre des objets avant et après la simulation. Finalement, le module de post-traitement permet d'exporter les résultats de la simulation pour une utilisation ultérieure.

7.2 Objectifs de la connexion

La plate-forme SALOME permet de déployer des applications sur des ressources qui sont spécifiées dans une liste statique. Nous souhaitons faire une proposition permettant à SALOME de déployer des applications sur les ressources d'une grille dynamique, de façon transparente pour les utilisateurs. Nous détaillons dans cette partie les objectifs que nous avons souhaités atteindre.

7.2.1 Permettre à SALOME de profiter simplement de la puissance d'une grille

La plate-forme SALOME repose sur une architecture distribuée. Ainsi, les composants de la plate-forme et les applications exécutées peuvent communiquer via un bus CORBA. Toutefois, les applications ne peuvent pas être déployées simplement sur plusieurs machines. En effet, la liste des machines et leurs caractéristiques doivent être renseignées au chargement de la plate-forme.

L'intérêt d'utiliser un système comme Vigne est dans ce cas de permettre à SALOME d'utiliser les ressources de calcul d'une grille sans que la plate-forme ne se préoccupe de la gestion des ressources. La sélection des ressources et le support de l'ajout/retrait des ressources de la grille sont complètement pris en charge par Vigne. Une application distribuée pilotée par SALOME pourrait simplement être exécutée sans modification de son code sur une grille composée de plusieurs milliers de ressources.

7.2.2 Simplifier le déploiement des applications

Lorsqu'une application distribuée est exécutée dans SALOME, toutes les ressources utilisées doivent posséder une installation de SALOME. Cette installation comprend un certain nombre de librairies logicielles et de variables d'environnement correctement positionnées. Installer SALOME sur tous les nœuds d'une grille peut sembler contraignant et même impensable si le nombre de ressources est grand. En effet, installer et maintenir à jour les ressources pour que toutes possèdent la dernière version de la plate-forme est d'une complexité qui augmente

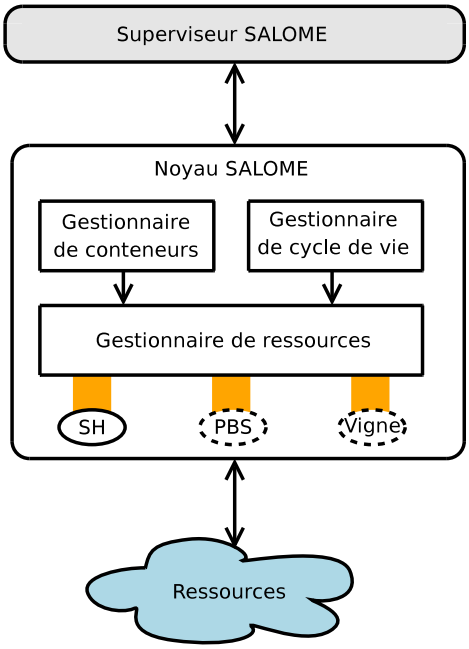


FIG. 7.3 – Architecture du noyau de SALOME

avec la taille de la grille. Une solution est d'utiliser un système de fichier distribué de type NFS pour éviter l'installation sur chaque nœud de la grille. Toutefois, l'utilisation à l'échelle d'une grille d'un système comme NFS n'est pas envisageable compte-tenu de son architecture centralisée.

Nous souhaitons donc simplifier le déploiement des applications sur des ressources distribuées pour que l'installation de SALOME sur les nœuds de calcul ne soit pas requise.

7.2.3 Exécuter les applications de façon fiable

Nous avons vu dans la partie 5.2 que Vigne permettait d'exécuter de façon fiable des applications en dépit de la volatilité des ressources qui composent une grille.

En utilisant Vigne pour gérer le déploiement des composants d'un schéma de calcul, SALOME peut profiter des propriétés de Vigne qui garantissent que l'exécution d'une application atteindra son terme si le code est dépourvu d'erreur et si suffisamment de ressources de calcul sont présentes.

7.3 Architecture modulaire de SALOME pour la connexion à un gestionnaire de ressources

Le noyau de SALOME a été conçu de façon à abstraire l'accès aux ressources. La figure 7.3 présente son architecture pour ce qui concerne l'exécution d'applications développées selon le

modèle de composant de SALOME [142]. Dans ce modèle, chaque tâche de l'application est un composant.

Il faut savoir qu'avec SALOME, les composants sont exécutés dans des *conteneurs*. Un conteneur est un serveur de composant qui permet de faire le lien entre SALOME et un ensemble de composants.

Pour expliquer l'architecture du noyau, voyons comment se passe l'exécution d'une application dans SALOME. Tout d'abord, le module de supervision exécute le schéma de calcul en utilisant le gestionnaire de cycle de vie. Le gestionnaire de cycle de vie demande au gestionnaire de conteneurs de déployer des conteneurs sur les ressources qui seront utilisées pour exécuter l'application. Le déploiement d'un conteneur est effectué par le gestionnaire de ressource adéquat, qui est choisi en fonction de la ressource utilisée. Les conteneurs sont déployés lorsque des tâches de l'application sont prêtes à être exécutées. En effet, si l'application est représentée par un *workflow*, toutes les tâches ne peuvent pas être exécutées simultanément, ainsi tous les conteneurs ne sont pas déployés dès le début. Lorsqu'un conteneur est déployé, un composant, sous la forme d'une librairie dynamique, y est chargé et son démarrage est initié par le module de supervision.

Ainsi, le gestionnaire de ressources du noyau permet à SALOME de s'adapter à divers types de ressources comme des ressources exploitées en mode interactif ou avec un gestionnaire de traitement par lots. Actuellement, SALOME permet d'exécuter des applications sur des ressources exploitées en mode interactif et dispose d'un support expérimental pour l'utilisation de ressources exploitées avec un gestionnaire de traitement par lots.

C'est grâce à cette architecture modulaire que nous avons réalisé la connexion de SALOME au système Vigne. Pour exploiter une grille dont les ressources sont fédérées par Vigne, SALOME utilise un gestionnaire de ressources spécialisé pour Vigne.

7.4 Spécialisation du gestionnaire de ressources de SALOME pour la connexion à Vigne

Comme nous l'avons indiqué dans la partie 7.3, ajouter le support pour un nouveau type de ressources dans SALOME revient à faire une nouvelle spécialisation du gestionnaire de ressources.

Dans cette partie, nous étudions les opérations devant être effectuées lorsque SALOME doit exécuter une tâche applicative, un composant au sens SALOME, pour le cas où des ressources exploitées en mode interactif sont utilisées ou pour les cas où des ressources exploitées par Vigne sont utilisées.

Lorsqu'un composant SALOME est prêt à être exécuté, le superviseur vérifie si un conteneur déjà déployé est libre pour lancer ce composant. Nous étudions tout d'abord le cas où aucun conteneur n'est disponible, puis, le cas où un conteneur est libre.

Pour simplifier les explications, nous présentons uniquement le cas de l'exécution des tâches non parallèles.

7.4.1 Chargement d'un composant dans un nouveau conteneur

7.4.1.1 Ressources exploitées en mode interactif

Le conteneur doit être déployé sur la ressource choisie avec une commande d'exécution à distance de type `ssh`. Comme une installation de SALOME est supposée uniforme sur tous les nœuds, éventuellement grâce à NFS, aucun transfert explicite de fichier n'est nécessaire.

Une fois le conteneur déployé, le gestionnaire de cycle de vie de SALOME lui demande le chargement du composant, qui lui aussi est présent sur tous les nœuds. D'un point de vue système, ce chargement est effectué par l'appel à la méthode `dlopen()` qui permet de charger une libraire dynamique en mémoire.

7.4.1.2 Ressources exploitées avec Vigne

Si les ressources sont exploitées avec Vigne, il faut distinguer le cas où le composant possède des dépendances avec d'autres composants ou non.

Cas des tâches sans dépendances Le gestionnaire de cycle de vie de SALOME ne possède pas la notion d'application mais uniquement la notion de tâches. Ainsi, lorsque le superviseur commence à exécuter un *workflow*, nous avons supposé dans notre spécialisation du gestionnaire de ressources pour Vigne que la description de l'application était fournie par le superviseur.

Cette description comprend le *workflow* de l'application et le chemin vers les binaires des tâches de l'application. Nous transformons cette description en une requête Vigne à l'aide d'un compilateur développé spécifiquement. Le *workflow* est traduit en un ensemble de dépendances spatiales, de synchronisation et de précédence, comme nous l'avons défini dans la partie 5.1.3.2.

Les dépendances de précédence ne sont pas exprimées comme dans le cas où un *workflow* est exécuté dans Vigne. En effet, le modèle de programmation de SALOME impose que les tâches soient lancées dans un conteneur. D'un point de vue système, Vigne détecte le premier processus lancé sur la ressource. Cela pose un problème puisque ce premier processus correspond à celui du conteneur et non à celui de la tâche qui n'est qu'une libraire dynamique. À la fin de l'exécution d'une tâche, le conteneur n'est pas détruit car dans la philosophie SALOME, il peut servir à l'exécution d'une autre tâche. Ainsi, Vigne n'est pas capable de détecter la fin de l'exécution d'une tâche car le processus du conteneur est toujours présent. L'enchaînement des tâches du *workflow* ne peut donc pas être initié par le moteur interne de Vigne mais il doit l'être de façon externe, en l'occurrence par le module de supervision. Pour cette raison, les dépendances de données entre tâches sont spécifiées comme externes, c'est-à-dire que pour qu'une tâche dépendante du résultat d'une autre tâche soit exécutée, il faut une action externe à Vigne.

Une fois la requête Vigne créée, le gestionnaire de ressource spécialisé la soumet à Vigne et enregistre l'identifiant de l'application Vigne. Lorsque des tâches peuvent être exécutées, c'est-à-dire si elles ne possèdent pas de dépendances de précédence, le gestionnaire de ressource demande le déploiement des conteneurs pour ces tâches.

Pour chaque tâche pouvant être exécutée, le traitement suivant est réalisé. Vigne découvre un nœud pouvant exécuter un conteneur et détecte toutes les dépendances, en termes de librairies dynamiques, nécessaires à l'exécution du conteneur et du composant. Ensuite, une archive contenant le conteneur, le composant et toutes les dépendances, est créée puis envoyée sur la ressource qui a été choisie par Vigne. Un script capable d'exécuter le conteneur avec les paramètres lui permettant de se connecter au gestionnaire de cycle de vie est également envoyé sur la ressource choisie. Une fois réceptionné, le script est exécuté par Vigne. Le script positionne un certain nombre de variables d'environnement, décompresse l'archive et exécute le conteneur. Ensuite, le composant est chargé comme dans le cas où les ressources sont utilisées en mode interactif.

Cas des tâches avec dépendances Dans ce cas la requête Vigne et les archives n'ont pas besoin d'être créées. Le gestionnaire de ressource spécialisé notifie Vigne pour que le flot d'exécution qui était mis en attente suite à l'expression d'une dépendance à une autre tâche puisse se poursuivre. Cette notification est envoyée avec le numéro d'application qui avait été enregistré par le gestionnaire de ressources et le numéro du composant à exécuter.

Sur réception de la notification, Vigne initie une découverte de ressources pour le composant concerné. Lorsque la découverte de ressources est terminée, le gestionnaire de ressources spécialisé pour Vigne envoie l'archive incluant un conteneur et le composant devant être exécuté sur la ressources choisie. Ensuite, les opérations sont identiques au cas précédent.

7.4.2 Chargement d'un composant dans un conteneur existant

7.4.2.1 Ressources exploitées en mode interactif

Lorsque les ressources sont exploitées en mode interactif SALOME se contente de demander le chargement du composant dans un conteneur existant et d'invoquer la méthode principale de démarrage du composant.

7.4.2.2 Ressources exploitées avec Vigne

Lorsqu'un conteneur doit être déployé, Vigne effectue une soumission de tâche. Cela sous-entend qu'une découverte et une allocation de ressources sont effectuées. Si le conteneur existe déjà, il n'est pas nécessaire de découvrir et d'allouer de nouvelles ressources. Ainsi, du point de vue de Vigne, cela est vu comme une extension de tâche existante.

Ainsi, la primitive de chargement de composant crée une archive contenant le composant à charger ainsi que toutes les librairies dont il dépend. Cette archive est envoyée sur le nœud qui contient déjà le conteneur, ainsi qu'un script qui sera capable de décompresser l'archive. Une fois l'archive et le script reçus, le script est exécuté. Ensuite le gestionnaire de cycle de vie demande au conteneur le chargement du composant et invoque la méthode principale de démarrage de composant.

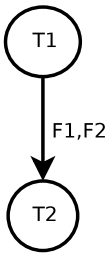


FIG. 7.4 – *Workflow* simple composé de deux tâches

7.4.3 Synthèse des fonctionnalités ajoutées à Vigne

Afin de piloter l'exécution d'une application depuis un moteur externe comme SALOME, certaines fonctionnalités de Vigne doivent être inhibées et d'autres, étendues.

Le moteur interne de *workflow* doit être inhibé pour ne pas enchaîner automatiquement les tâches de l'application car dans le cas des applications SALOME, il n'est pas possible de détecter la terminaison des tâches du point de vue système car les conteneurs peuvent être persistants.

Afin de permettre au moteur externe de piloter l'enchaînement des tâches, il a été nécessaire d'ajouter un nouveau type de dépendance qui impose au moteur interne de Vigne de se mettre en attente. Nous avons donc ajouté une fonctionnalité à Vigne permettant à un moteur externe de notifier une tâche en attente d'une dépendance externe afin qu'elle puisse être exécutée.

Pour gérer le cas où un composant doit être chargé dans un conteneur existant, nous avons ajouté une fonctionnalité permettant d'étendre une tâche, au sens Vigne, en cours d'exécution. Cette extension est une notion générale pour l'exécution d'applications qui consiste en l'envoi de fichiers et en l'exécution d'un script. Appliquée à SALOME, l'extension de tâche consiste à envoyer une archive contenant le composant et ses dépendances sur la ressource utilisée par le conteneur et à exécuter un script capable de décompresser cette archive.

7.4.4 Interactions entre SALOME et Vigne à l'aide d'une exemple simple

Dans ce paragraphe, nous détaillons les interactions entre SALOME et Vigne qui se produisent lorsqu'une application est exécutée. Pour cela, nous prenons une application très simple qui est un *workflow* composé de deux tâches T1 et T2. Comme le montre la figure 7.4, T2 doit attendre la fin de l'exécution de T1 car elle a besoin des fichiers F1 et F2 qui sont produits en fin d'exécution de T1. De plus, nous supposons que T1 nécessite une ressource de type 1 pour s'exécuter et que T2 nécessite une ressource de type 2.

La figure 7.5 présente les interactions entre SALOME et Vigne lorsque l'application exemple est soumise.

Suite à la demande d'exécution de l'application effectuée par un utilisateur depuis la plate-forme SALOME, une demande est envoyée au superviseur. À partir du schéma de l'application qui a été enregistré dans la plate-forme, le superviseur transmet au noyau une demande de

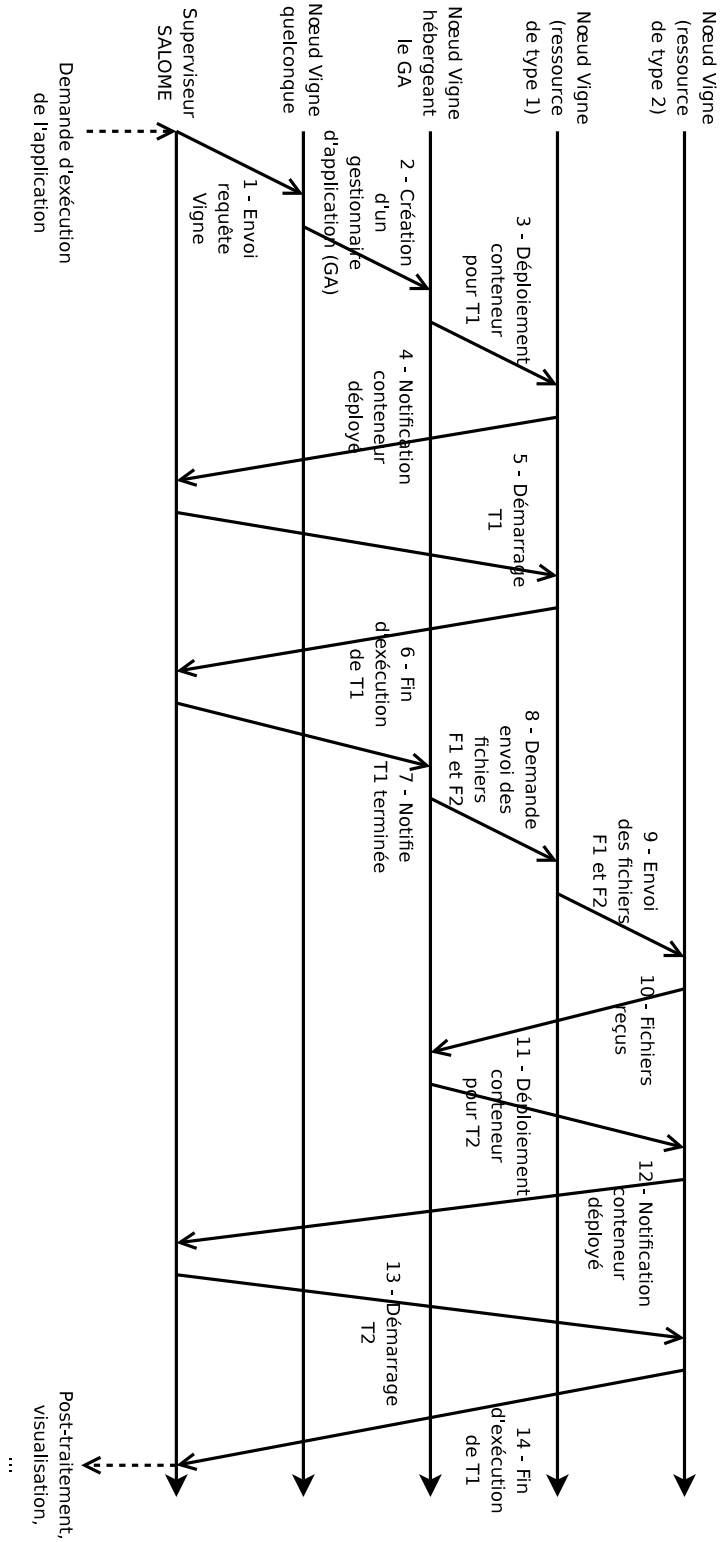


FIG. 7.5 – Schéma temporel des interactions entre SALOME et Vigne pour l'exécution d'un *workflow* simple

soumission d'application. Le noyau et plus particulièrement le gestionnaire de ressources spécialisé pour Vigne génère une requête Vigne et pour chaque tâche, une archive qui contient un conteneur, une librairie dynamique représentant la tâche et les dépendances associées. Les interactions entre Vigne et SALOME commencent à partir de ce moment.

La requête Vigne est soumise à n'importe quel nœud de la grille qui héberge un démon Vigne (flèche 1). Sur réception de cette requête, Vigne crée un gestionnaire d'application sur la grille (flèche 2). Une fois créé, le gestionnaire d'application recherche des ressources pour toutes les tâches qui ne dépendent pas du résultat d'autre tâches, en l'occurrence pour T1. Lorsqu'un nœud convenant à l'exécution de T1 est trouvée, Vigne déploie le conteneur de T1 (flèche 3). Le conteneur notifie le superviseur de SALOME de son lancement (flèche 4) et ce dernier demande l'exécution de la tâche (flèche 5). Pour mémoire, l'exécution de la tâche se résume au chargement d'une libraire dynamique dans le conteneur déjà déployé. À la fin de l'exécution de T1, le superviseur est notifié (flèche 6). À ce moment là, le superviseur notifie Vigne qui était en attente pour que la tâche T2 soit exécutée (flèche 7). Vigne recherche un nœud pouvant exécuter T2 et lorsqu'il est trouvé, le système demande au nœud qui hébergeait T1 d'envoyer les fichiers de sorties F1 et F2 sur le nœud choisi pour T2 (flèche 8). Sur réception de cette demande, le nœud utilisé pour T1 envoie F1 et F2 sur le nœud choisi pour T2 (flèche 9). Une fois les fichiers reçus, le nœud choisi pour T2 notifie le gestionnaire d'application (flèche 10). Vigne déploie alors le conteneur pour T2 (flèche 11). Une fois déployé, le conteneur notifie SALOME (flèche 12) qui demande l'exécution de T2 (flèche 13). À la fin de l'exécution de T2, le superviseur est notifié (flèche 14) et les opérations de post-traitement ou de visualisation peuvent être effectuées par l'utilisateur.

7.5 Évaluation avec une application industrielle

Nous avons évalué notre proposition avec une application industrielle utilisée et développée à EDF R&D. Cette application est un couplage entre le code de mécanique des fluides Code_Saturne et le code de thermique solide Syrthes. Le couplage simule un phénomène de transfert de chaleur en milieu turbulent.

L'objectif est d'exécuter cette simulation numérique depuis la plate-forme SALOME en utilisant les ressources d'une grille exploitée par le système Vigne.

Nous présentons dans cette partie l'architecture de l'application utilisée pour l'évaluation, les fonctionnalités de Vigne qui ont été sollicitées et le résultat d'expérimentations qui ont été conduites pour valider l'approche.

7.5.1 Architecture de l'application Saturne/Syrthes

Nous avons vu que l'application était composée du couplage de Code_Saturne et de Syrthes. La figure 7.6 montre l'architecture de cette application.

L'application comporte trois phases de calcul. Tout d'abord, un module de pré-traitement divise le maillage initial pour Code_Saturne en n maillages, n étant le nombre de nœuds qui sont utilisés par la suite pour le code parallèle Code_Saturne. Ensuite, Code_Saturne est exécuté sur n nœuds, en parallèle de Syrthes. En fonction du nombre de pas de temps de la simulation, un nombre variable d'itérations est effectué. À chaque itération, Code_Saturne effectue un calcul

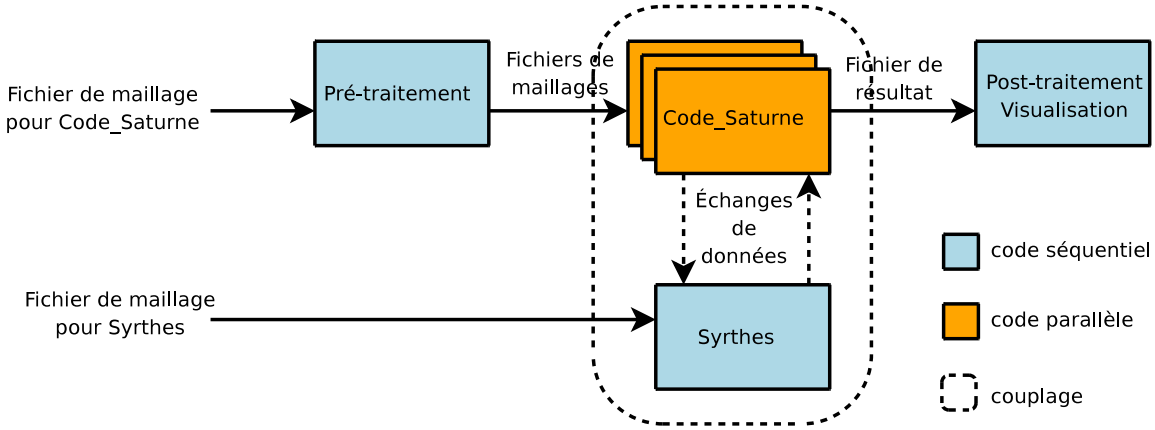


FIG. 7.6 – Architecture de l'application de couplage Code_Saturne/Syrthes

et envoie le résultat à Syrthes qui effectue un calcul et qui retourne le résultat à Code_Saturne. Une fois les itérations achevées, le résultat, sous la forme d'un fichier MED, est envoyé au module de post-traitement afin de visualiser les résultats de la simulation.

7.5.2 Fonctionnalités sollicitées de Vigne

Dans cette partie, nous présentons les fonctionnalités de Vigne qui sont sollicitées par l'exécution de cette application industrielle à travers la plate-forme SALOME.

Les ressources sont exploitées de la même façon que lorsque des applications sont soumises directement à Vigne, c'est-à-dire que c'est l'interface avec les ressources qui est utilisée. Via sa connexion à Vigne, SALOME peut donc profiter de ressources exploitées de façon interactive avec Linux ou Kerrighed et de ressources exploitées avec un gestionnaire de traitement par lots comme Torque.

La plate-forme SALOME ne dialogue pas directement avec toutes les ressources de la grille comme c'est le cas par défaut. Vigne permet de trouver automatiquement des ressources adaptées à l'exécution des applications grâce à ses fonctionnalités de découverte de ressources. De plus les ressources sont choisies grâce au service d'allocation de ressources. Dans le cas de l'application testée, la partie Code_Saturne est parallèle. Ainsi, pour une exécution efficace, les tâches parallèles doivent être exécutées sur des ressources ayant une connectivité réseau efficace. La fonction de co-allocation de ressources sur des ressources proches est donc utilisée.

L'application de test est composée de tâches possédant tous les types de dépendance qui sont définis dans Vigne. En effet, le *workflow* induit des dépendances de précédence entre les tâches, le couplage entre Syrthes et Code_Saturne induit une dépendance de synchronisation puisque les deux composants doivent être exécutés simultanément. Finalement, Code_Saturne étant une application parallèle, cela fait intervenir des dépendances spatiales et de synchronisation.

Les fonctionnalités de surveillance d'applications sont également sollicitées puisque toute défaillance de ressource ou de tâche est reportée. Sachant que des flux de données peuvent

idApplication **VigneSoumissionApplication**(*descApplication*)
int **VigneSoumissionExtensionTache**(*idApplication, idTache, descApplication*)
int **VigneNotifierDemarrageTache**(*idApplication, idTache*)
int **VigneTransfertFichiersEntree**(*idApplication, listeFichiers*)
int **VigneTransfertFichiersSortie**(*idApplication, listeFichiers*)
int **VigneTransfertFichiersEntreTaches**(*idApplication, idT1, idT2, listeFichiers*)

FIG. 7.7 – Interface utilisée par SALOME pour le pilotage externe d'applications avec Vigne

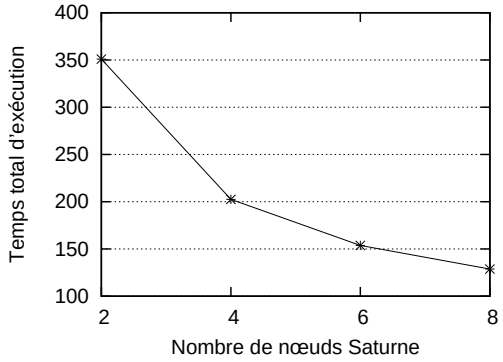


FIG. 7.8 – Temps d'exécution de l'application Code_Saturne/Syrthes pour 3 itérations de calcul

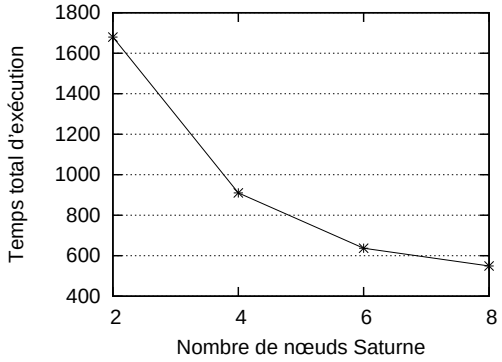


FIG. 7.9 – Temps d'exécution de l'application Code_Saturne/Syrthes pour 30 itérations de calcul

être échangés entre diverses tâches d'une application SALOME sans que Vigne ne puisse en être informé et que la reconnexion d'un conteneur SALOME ne peut être effectuée sans une instruction du module de supervision, la prise de décision suite à la détection de défaillance ne peut pas être effectuée par Vigne, mais par le module de supervision de SALOME. Ainsi, la fonctionnalité de redémarrage de tâche de Vigne en cas de défaillance ne peut pas être utilisée. Les défaillances sont cependant reportées au module de supervision de SALOME.

Afin de simuler la transparence d'utilisation d'un système de fichier distribué, les fonctionnalités d'échanges de fichiers de Vigne entre les tâches et d'envoi et récupération des fichiers d'entrée/sortie sont utilisées.

Finalement, l'interface pour la connexion d'un moteur externe de pilotage à Vigne a été utilisée. Les fonctions utilisées sont présentées dans la figure 7.7.

7.5.3 Expérimentation

L'expérimentation a consisté en l'exécution de l'application couplée Code_Saturne/Syrthes sur une grappe d'EDF R&D.

Tout d'abord, nous nous sommes assurés que l'application s'exécutait correctement lorsque les ressources de la grappe étaient fédérées par Vigne. Ensuite, nous avons vérifié que les ressources étaient correctement allouées par Vigne. Pour cela, nous avons vérifié que le temps d'exécution de l'application diminuait avec l'augmentation du nombre de nœuds de la même

façon en utilisant les ressources fédérées par Vigne qu'en utilisant les ressources directement.

Nous avons fait varier le nombre de nœuds utilisés pour le code parallèle Code_Saturne entre 2 et 8 et nous avons effectué deux exécutions. Dans la première nous avons calculé 3 itérations et dans la seconde nous avons calculé 30 itérations.

À EDF R&D, les grappes de production sont exploitées avec un gestionnaire de travaux et les nœuds de chaque grappe sont dans un réseau privé. Actuellement, Vigne possède une limitation qui empêche l'utilisation de nœuds situés dans des réseaux privés car leurs adresses IP ne sont pas routables vers d'autres réseaux privés. Pour ces raisons, nous avons utilisé les nœuds d'une seule grappe d'expérimentation.

Pour réaliser cette expérimentation, nous avons utilisé 12 nœuds de type bi-processeurs Intel Xeon cadencés à 3.4 GHz et dotés de 4 Go de mémoire vive. `vigne_d` a été déployé sur chacun des 12 nœuds et SALOME a été lancé sur l'un des nœuds.

Les figures 7.8 et 7.9 montrent respectivement le résultat des exécutions pour 3 et 30 itérations. Ces résultats montrent que le temps d'exécution de la partie parallèle de l'application, c'est-à-dire Code_Saturne, diminue avec l'augmentation du nombre de nœuds utilisés. L'accélération mesurée est strictement identique à celle mesurée sans utiliser le système Vigne. Par conséquent, cela démontre l'efficacité du placement des tâches sur les ressources effectué par l'allocateur de ressources de Vigne.

Compte-tenu du faible nombre de nœuds utilisés, cette expérience n'est pas représentative d'une utilisation sur une grille et encore moins d'une grille dynamique. Toutefois, cette expérience est une preuve de concept puisque le déploiement de l'application a été réalisé uniquement avec Vigne, sans utiliser les fonctionnalités habituellement utilisées sur une grappe comme un système de fichier distribué de type NFS et sans avoir à déclarer l'ensemble des ressources à la plate-forme SALOME. De plus, l'utilisateur n'a pas besoin d'effectuer plus d'opérations que s'il utilisait SALOME avec une seule machine.

7.6 Conclusion

Nous avons présenté dans ce chapitre les travaux que nous avons effectués pour connecter la plate-forme industrielle de simulation numérique SALOME au système Vigne. Nous avons tout d'abord présenté la plate-forme SALOME en détaillant ses fonctionnalités et son architecture. Nous avons ensuite présenté les motivations et les objectifs de la connexion de SALOME à Vigne. Puis nous avons présenté notre approche consistant à étendre le noyau de SALOME. Nous avons également présenté une évaluation avec l'application industrielle Code_Saturne/Syrthes.

Ces travaux nous permettent de tirer plusieurs enseignements.

Tout d'abord, cela nous a permis d'évaluer le système Vigne avec une application industrielle de couplage de codes sous la forme d'un *worklfow*. Ce type d'application fait intervenir l'ensemble des fonctionnalités de Vigne qui ont été présentées dans les chapitres 3, 4 et 5 de cette thèse et les fonctionnalités intégrées dans l'extension du noyau SALOME que nous avons développée.

Ensuite, nous avons conçu le système Vigne de façon à ce que les applications puissent être pilotées de deux façons. La première façon qui utilise un moteur de pilotage interne à Vigne

permet aux utilisateurs d'exécuter directement des applications décrites sous la forme d'un *workflow* simple. La seconde façon permet d'utiliser Vigne avec un moteur externe pour le pilotage des applications. Nos travaux sur la connexion de SALOME à Vigne nous ont permis de montrer la faisabilité du concept faisant cohabiter deux modes de pilotage des applications avec une plate-forme de simulation numérique utilisée en milieu industriel.

Nous avons également montré que l'utilisation d'un système comme Vigne pouvait simplifier de façon significative l'accès aux ressources d'une grille. En effet, la plate-forme SALOME, moyennant une simple extension de son gestionnaire de ressources comme cela est présenté dans la partie 7.4, peut bénéficier de toutes les fonctionnalités de Vigne qui simplifient l'accès à la grille. Par exemple, doté de cette extension, SALOME peut utiliser les ressources de la grille sans connaître leur localisation. L'évolution des ressources dans la grille ne nécessite pas non plus d'opération d'administration sur SALOME.

Grâce à des mécanismes de détection des dépendances système auxquels les composants applicatifs sont liés et grâce à un système d'empaquetage de ces dépendances, nous avons montré qu'il était simple de supprimer le lourd pré-requis imposant une installation de SALOME sur toutes les ressources de la grille. Cette idée nous a permis de montrer que le déploiement d'applications pouvait très bien s'affranchir de la présence des librairies dynamiques avec lesquelles sont liés les composants applicatifs. Cette solution peut en partie résoudre les problèmes d'uniformité d'installation du système d'exploitation des ressources d'une grille. Un problème non résolu reste celui de l'hétérogénéité des architectures qui empêche l'utilisation de toutes les ressources hétérogènes, du point de vue des architectures processeur, de la grille. Des solutions fondées sur des machines virtuelles pourraient résoudre ce problème.

Sans prétendre que le système Vigne possède la robustesse d'un système industriel, l'évaluation avec une application industrielle patrimoniale nous a permis de valider l'intérêt pragmatique des concepts que nous avons proposés dans cette thèse.

Plusieurs perspectives sont envisageables pour poursuivre ce travail.

Il serait intéressant d'étudier les interactions entre le module de supervision de SALOME et Vigne afin de gérer les défaillances lorsqu'elles sont reportées par Vigne.

Nous avons supposé que le module de supervision de SALOME fournissait une description de la topologie de l'application que nous pouvions utiliser comme base pour générer une requête Vigne. Actuellement, cette description n'est pas fournie automatiquement et une étude devrait être menée en collaboration avec l'équipe SALOME pour spécifier une description suffisamment générique qui permettrait d'exécuter à travers Vigne tous types d'applications, y compris ceux décrits par un *workflow* possédant des cycles comme cela peut être défini dans SALOME.

Pour profiter de l'ensemble des grappes d'EDF R&D, il serait nécessaire de supprimer la limitation de Vigne concernant l'utilisation de ressources situées dans des réseaux privés.

Afin de rendre possible le déploiement de composants sans que les ressources ne possèdent d'installation de SALOME, notre solution consiste à empaqueter toutes les dépendances des conteneurs et des composants et à les envoyer sur les nœuds utilisés. Afin de réduire la bande passante consommée pour cela, il serait intéressant de comparer les dépendances nécessaires avec celle déjà présentes sur les nœuds cibles afin de n'envoyer que celles qui sont nécessaires.

Conclusion et perspectives

Conclusion générale

La simulation numérique est un champ d'application de l'informatique qui croît sans cesse et qui est largement utilisé dans de nombreux domaines scientifiques. Afin de modéliser des phénomènes toujours plus complexes et de disposer de résultats d'une précision croissante, d'importants moyens de calculs doivent être utilisés.

Les grilles de calcul sont une cible de choix pour les applications de simulation numérique compte-tenu de la puissance de calcul qu'elles offrent. Toutefois les ressources d'une grille sont plus complexes à utiliser que des stations de travail ou des grappes de calculateurs. Cela est dû à leur volatilité, à leur hétérogénéité et à la grande échelle qui empêche les utilisateurs d'avoir une connaissance globale de l'ensemble des ressources.

De nombreux projets ont été menés dans le contexte de l'utilisation des grilles. Ces projets ont souvent été mis en œuvre sous la forme d'intergiciels de communication, de gestion globale des ressources ou encore de support à la programmation d'applications pour grilles.

Une étude comparative des caractéristiques des projets existants nous a permis de tirer un ensemble de leçons. Tout d'abord, un certain nombre d'intergiciels sont fondés sur une architecture centralisée ou hiérarchique qui peut s'avérer être un frein au passage à l'échelle et qui est sensible aux défaillances. Ensuite, les intergiciels existants ne sont pas forcément adaptés à tous types d'applications et lorsqu'ils le sont, ils sont très complexes à utiliser. De plus, pour proposer des fonctionnalités avancées comme la co-allocation de ressources, les intergiciels de l'état de l'art ne supportent pas forcément les applications patrimoniales et peuvent nécessiter une ré-édition de liens, un ajout de primitives spécifiques dans le code ou encore une conception spécialisée. Finalement, les intergiciels existants ne sont pas dotés de mécanismes permettent la détection fine des défaillances d'applications sans modification des codes. De plus, peu d'intergiciels sont capables de garantir que dans une grille composée de ressources volatiles, une application exécutée atteindra le terme de son exécution si elle ne comporte pas d'erreurs de programmation.

Contributions

L'ensemble de nos contributions s'inscrit dans la conception d'un système permettant de gérer les ressources des grilles. Notre approche vise à traiter les questions liées à :

- la grande échelle d'une grille ;
- la volatilité de ses ressources ;

– la simplicité, la fiabilité et l'efficacité d'exécution des applications distribuées et patrimoniales ;
– la simplicité d'administration.

Dans la suite, nous synthétisons les cinq contributions majeures de cette thèse puis la mise en œuvre et l'évaluation de ces contributions dans un prototype fonctionnel.

Architecture complètement distribuée pour la gestion des ressources d'une grille de très grande taille

Nous avons proposé une architecture permettant de gérer les ressources d'une grille. Cette architecture est composée d'un ensemble de services de niveau système qui permettent à des utilisateurs d'exécuter des applications distribuées sur une grille composée de ressources hétérogènes. L'architecture du système a été présentée dans [100] et [103].

Tous les services du système sont distribués afin de ne créer aucun point de congestion ou aucun point critique dont la défaillance compromettrait tout le système. De plus, la distribution permet de supporter une utilisation avec une grille composée de plusieurs milliers de ressources.

La distribution du système est complètement occultée aux utilisateurs grâce à une vision de type *système à image unique*. Cela repose principalement sur l'utilisation des réseaux pair-à-pair structurés qui permettent d'abstraire la localisation des services du système. Un utilisateur peut donc accéder à l'ensemble des services du système depuis n'importe quelle ressource de la grille. Les fondations pair-à-pair du système ont été présentées dans [104].

Finalement, tous les services du système ont été conçus de façon à ce que la défaillance de n'importe quelle ressource de la grille ne mette pas en péril le reste du système. Par conséquent, notre approche ne suppose l'existence d'aucune entité stable dans la grille.

Système d'information distribué pour les grilles de grande échelle

Nous avons proposé un système d'information distribué adapté à des grilles de très grande échelle. Il permet de découvrir des ressources dans la grille à partir d'une description composée d'un ensemble de critères qui représentent les pré-requis matériels et logiciels pour l'exécution d'une application.

Ce service repose sur l'utilisation de réseaux pair-à-pair non structurés et sur le concept de marches aléatoires. Cette approche permet de s'affranchir d'une solution de type annuaire, fréquemment utilisée dans l'état de l'art, qui pose une limite au passage à l'échelle.

L'utilisation d'un réseau pair-à-pair rend le système d'information auto-organisant. En effet, l'ajout de ressources dans la grille se résume à une opération locale d'insertion dans le système et la suppression d'une ressource ne requiert aucune opération.

L'évolution des caractéristiques d'une ressource ne nécessite aucune intervention de l'administrateur. Comme le reste de l'architecture du système, le système d'information est tolérant à la défaillance de n'importe quel élément de la grille grâce aux propriétés d'auto-réparation du réseau pair-à-pair non structuré.

Nous avons proposé une optimisation du protocole de marches aléatoires afin d'en augmenter l'efficacité dans le cadre d'utilisation des grilles. Le concept repose sur l'utilisation de

caches dotés d'une politique favorisant la découverte de ressources libres et sur la dissémination d'une partie des informations découvertes dans le réseau pair-à-pair non-structuré. Cette optimisation du protocole de marches aléatoires a été partiellement présentée dans [101].

Nous avons également proposé un mécanisme permettant d'augmenter les chances de localiser les ressources rares d'une grille. Ce mécanisme permet d'évaluer la rareté de chaque ressource et de faire connaître celles qui sont rares.

Le système d'information que nous avons proposé est un service dont les propriétés sont nécessaires à la conciliation des éléments suivants : la grande échelle d'une grille, la volatilité des ressources et la simplicité d'administration. Ce service contribue aussi à la simplicité d'exécution d'applications sur la grille puisqu'il affranchit les utilisateurs de la recherche de ressources disponibles et du besoin de connaître globalement les ressources de la grille.

Allocation et co-allocation efficace des ressources dans une grille

Nous avons proposé un service d'allocation de ressources générique permettant de mettre en œuvre de nombreuses politiques d'allocation de ressources.

Ce service permet également de co-allouer des ressources pour des applications parallèles ou distribuées composées de tâches devant s'exécuter de façon synchronisée et sur des nœuds possédant une interconnexion réseau efficace. Les affinités entre les tâches d'une application peuvent être décrites par l'utilisateur grâce à un formalisme simple que nous avons défini. Les mécanismes de co-allocation de ressources ont été présentés dans [102].

Ces fonctionnalités contribuent à la simplicité et à l'efficacité de l'exécution d'applications sur une grille.

De plus, ce service ne requiert aucune modification des codes applicatifs afin de supporter les applications patrimoniales.

Exécution fiable d'applications d'applications distribuées

Nous avons proposé un service de gestion d'application permettant de prendre en charge le cycle de vie complet des applications distribuées. En particulier, ce service repose sur le système d'information et sur le service d'allocation de ressources afin de découvrir et d'allouer des ressources capables d'exécuter les tâches applicatives.

Le service de gestion d'application est complètement distribué. La gestion du cycle de vie de chaque application est potentiellement réalisée sur une ressource différente, choisie aléatoirement dans le système. Grâce à la vue SIU, les actions concernant une application comme une demande d'état ou une suppression d'application peuvent être initiées sans que l'utilisateur ait à connaître la localisation des informations liées à une application.

Le service de gestion d'application intègre un moteur de *workflow* permettant de gérer les relations de précédence pouvant exister entre les tâches d'une application. Lorsque de telles relations existent, il est très probable que des données doivent transiter entre les tâches concernées. Si ces données sont des fichiers, le service de gestion d'application se charge de les transférer automatiquement entre les ressources.

Ce service assure que les applications puissent atteindre le terme de leur exécution en surveillant finement les tâches qui pourraient connaître une défaillance non liée au code appli-

catif et en détectant la disparition de nœuds utilisés par l'application. À la détection d'une défaillance, le service de gestion d'application est capable de redémarrer les tâches concernées en tenant compte des affinités que ces tâches ont avec d'autres et en redémarrant éventuellement des tâches n'ayant pas subi de défaillance. Les mécanismes de supervision d'application de Vigne ont été présentés dans [149].

Le service de gestion d'application simplifie considérablement l'exécution d'application sur une grille puisque il prend en charge les relations spatiales, de précédence et de synchronisation entre les tâches et pilote entièrement l'exécution des applications. De plus, ce service permet d'atteindre l'objectif de fiabilité d'exécution des applications. Le service de gestion d'application a été présenté dans [102].

Mise en œuvre d'un prototype fonctionnel et évaluation à grande échelle

Nous avons mis en œuvre un prototype fonctionnel permettant de valider nos contributions.

Différentes expérimentations ont été réalisées en conditions réelles d'exécution sur la plate-forme Grid'5000 où le système Vigne a été déployé en utilisant un grand nombre de processeurs sur plusieurs sites géographiques.

Une évaluation de l'infrastructure de communication nous a permis de vérifier que l'utilisation de réseaux logiques structurés et non-structurés est tout à fait adaptée pour une utilisation avec une grille de grande taille. En effet, cette infrastructure est une brique essentielle pour la construction d'un système permettant de concilier la vue SIU qui permet de masquer la distribution des ressources aux utilisateurs et les objectifs de très grande échelle.

L'évaluation des mécanismes d'allocation de ressources nous a permis de vérifier que l'utilisation d'un ordonnanceur complètement décentralisé permet d'obtenir des résultats presque similaires à ceux d'un ordonnanceur global centralisé pour ce qui concerne l'efficacité du placement des tâches.

L'évaluation des mécanismes de co-allocation de ressources montre qu'ils permettent d'améliorer de façon significative le temps d'exécution d'applications communicantes en tentant d'allouer des ressources garantissant une bonne connectivité réseau à des applications composées de tâches communicantes.

Nous avons évalué la capacité de notre système à exécuter de façon fiable des applications. Les résultats montrent qu'en dépit des défaillances qui ont été injectées lors de l'expérience, toutes les applications ont été exécutées correctement.

Finalement, nous avons évalué, en simulation, le protocole de découverte de ressources que nous avons proposé. Les résultats montrent que ce protocole est adapté à la construction d'un système d'information pour les grille de grande taille qui sont particulièrement chargées car il permet de trouver des ressources disponibles dans une grille même lorsqu'un grand nombre de ressources est déjà occupé, tout en étant économe en bande-passante.

Validation industrielle

L'approche que nous avons défendue dans cette thèse a été validée à EDF R&D dans le cadre du couplage de Vigne à la plate-forme de simulation numérique SALOME. Ces travaux

permettent à la plate-forme SALOME d'accéder simplement aux ressources d'une grille fédérées par Vigne et certaines hypothèses contraignantes de SALOME ont été levées comme par exemple la nécessité d'avoir un système de fichier distribué entre les ressources ou encore la présence d'une installation de la plate-forme sur chacune des ressources.

Une application industrielle, de type *workflow* et comportant un couplage de codes, a été exécutée à travers SALOME en utilisant des nœuds exploités par Vigne. À travers cette évaluation, tous les services de Vigne ont été sollicités, y compris l'interface permettant le pilotage d'une application depuis un moteur externe. Nous avons ainsi pu vérifier la faisabilité de notre approche et l'efficacité des services rendus par le système Vigne.

Applicabilité des résultats

Les contributions que nous avons proposées dans cette thèse peuvent être appliquées à la fédération des ressources volatiles des grilles de très grande taille.

Vigne est adapté à la volatilité des ressources d'une grille de type calcul global ou réseau de stations de travail. En effet, les connexions et déconnexions fréquentes des ressources mises à disposition dans la grille sont parfaitement supportées par les fondations pair-à-pair du système. De plus, le système garantit la fiabilité de l'exécution des applications en dépit de la volatilité des nœuds.

Vigne est également adapté pour des grilles de type fédérations de grappes ou les grilles hétérogènes où les réseaux d'interconnexion peuvent varier entre les ressources. En effet, le système permet d'optimiser l'exécution des applications composées de tâches communicantes grâce à des mécanismes de co-allocation de ressources qui maximisent le critère de localité, du point de vue du réseau, entre les ressources.

Nous avons conçu Vigne en souhaitant simplifier le plus possible l'utilisation des ressources d'une grille du point de vue des utilisateurs et aussi de celui des administrateurs. Cet intergiciel pourrait être utilisé dans des institutions où les utilisateurs n'ont pas de connaissances en informatique distribuée et où les administrateurs des moyens informatiques ne sont pas experts dans les systèmes distribués de très grande taille.

Perspectives

Au terme de cette thèse, nous envisageons un ensemble de perspectives de recherche à plus ou moins long terme.

Court terme

Évaluations étendues

Un premier axe de travail à court terme est lié à l'évaluation de nos concepts. Il serait intéressant d'évaluer plus finement le protocole de découverte de ressources proposé en faisant varier d'autres paramètres comme la profondeur d'une marche aléatoire ou encore la taille des caches.

Des évaluations des concepts de co-allocation de ressources et de gestion des applications distribuées pourraient être menées avec des applications industrielles. Une comparaison avec les mécanismes de co-allocation des intergiciels de l'état de l'art pourrait aussi être effectuée.

Finalement, il serait intéressant d'évaluer d'autres politiques d'allocations de ressources comme celles qui sont présentées dans la partie 4.3.2.

Améliorer la robustesse du prototype

L'implémentation que nous avons réalisée pour preuve de concept n'est pas suffisamment robuste pour une utilisation dans un contexte de production. En effet, le prototype n'est capable de gérer que les cas d'utilisation qui ont été prévus et une utilisation malicieuse du client Vigne pourrait conduire à une défaillance du système.

Améliorer l'ergonomie du prototype

Actuellement, il est nécessaire de fournir une description de l'application pour l'exécuter à travers Vigne. La réalisation de cette description peut être fastidieuse si l'application est composée de nombreuses tâches. Il pourrait être intéressant de concevoir une interface graphique pour le client Vigne afin de simplifier la génération des requêtes de soumission.

Moyen et long terme

Ordonnancement avec réservation de ressources

L'ordonnanceur que nous avons proposé ne nécessite pas de mécanisme de réservation de ressources. Il serait intéressant de faire évoluer le système Vigne pour que les ressources puissent être réservées. Cela permettrait de concevoir des politiques d'ordonnancement capables de répartir encore mieux les applications sur la grille en cas de forte charge.

Amélioration de la gestion des défaillances

Nous avons proposé un service de gestion d'application garantissant que l'exécution d'une application puisse atteindre son terme en dépit des défaillances des ressources. La stratégie basique que nous utilisons consiste à redémarrer les tâches ou les groupes de tâches concernés. Cette stratégie présente l'inconvénient majeur de perdre une partie des calculs ayant été effectués avant la défaillance. Il serait intéressant d'intégrer à Vigne un mécanisme permettant de prendre des points de reprises des tâches afin de les redémarrer depuis le dernier point de reprise en cas de défaillance. Afin de proposer un mécanisme transparent et générique pour les applications patrimoniales, il est nécessaire d'avoir un support du système d'exploitation. Le cas des applications composées de tâches non communicantes peut être traité de façon simple avec la capacité de point de reprise offerte par un système comme Kerrighed pour les nœuds de type grappe ou avec BLCR pour les nœuds de type calculateur sous Linux. Le cas des applications communicantes est plus délicat puisqu'il nécessite la sauvegarde des informations contenues dans les canaux de communication. Pour cela, aucune approche générique n'a été proposée actuellement. De plus, certains types de nœuds et certains types d'applications sont

capables de traiter d'eux-mêmes certains types de défaillance en prenant des actions correctives. Les mécanismes que nous avons mis en place dans le système de grille devraient donc être révisés pour tenir compte de la hiérarchie des mécanismes de support aux défaillances. Cette piste de recherche fait partie des travaux de thèse menés par Thomas Ropars, doctorant au sein de l'équipe-projet PARIS de l'INRIA.

Déploiement générique en tenant compte des modèles d'applications

Vigne offre une infrastructure système pour l'exécution d'applications. De fait, certaines opérations sont laissées à la charge des utilisateurs pour exécuter certains types d'applications comme les applications fondées sur le paradigme composant. Il est alors nécessaire d'effectuer des connexions spécifiques pour mettre en relation les différents composants. Ces opérations pourraient être réalisées via un script que l'utilisateur aurait écrit au préalable. Dans le cadre de l'extension que nous avons apportée à SALOME, nous avons mis en œuvre des fonctionnalités spécifiques pour supporter le déploiement et l'exécution d'applications fondées sur le modèle de composant de SALOME. Afin de soulager les utilisateurs de l'écriture de scripts spécifiques et plutôt que de développer spécifiquement des fonctionnalités pour chaque modèle d'application, il serait intéressant d'étudier le couplage de Vigne avec un outil de déploiement dédié. En l'occurrence, le projet Adage conçu au sein de l'équipe-projet PARIS de l'INRIA propose un outil de déploiement générique permettant de prendre en compte le modèle de programmation des applications pour le déploiement. De plus, une partie du travail de thèse de Boris Daix, doctorant au sein de l'équipe-projet PARIS de l'INRIA, apporte au projet Adage la notion de dynamicité dans le déploiement d'applications. Cette dynamicité permettrait d'étendre la fonctionnalité basique d'extension d'application actuellement présente dans Vigne afin de simplifier les opérations que l'utilisateur doit réaliser.

Authentification des utilisateurs

Dans un contexte de production, l'authentification des utilisateurs est nécessaire pour des raisons de sécurité, de journalisation ou de facturation. Cela dit, il est compliqué de gérer l'authentification d'utilisateurs pouvant provenir de plusieurs domaines d'administration et n'ayant pas forcément de compte sur tous les nœuds de la grille. Cela induit des opérations d'administration qui sont d'autant plus complexes que l'échelle de la grille est grande et que le nombre d'utilisateurs est important.

Globus [84] propose dans le module GSI [169] une authentification globale des utilisateurs grâce à un certificat X.509 qui est traduit sur chaque nœud de la grille vers une authentification locale qui correspond à un compte unique pour tous les utilisateurs. Comme tous les utilisateurs d'un même nœud possèdent le même compte local, un utilisateur malicieux peut perturber l'exécution des tâches des autres utilisateurs. Le projet XtreemOS [36] et en particulier le module *VO Management* propose une solution plus intéressante du point de vue de la sécurité puisque les compte globaux sont traduits vers un compte local dynamiquement créé. Ainsi, les exécutions de différents utilisateurs sont cloisonnées.

Utilisation de machines virtuelles

De nombreux travaux dans le domaine des machines virtuelles ont vu le jour comme par exemple Xen [50]. Ces nouveaux travaux permettent d'exécuter des applications avec un très faible ralentissement, de l'ordre de quelques pourcents seulement. L'utilisation de machines virtuelles sur les ressources d'une grille Vigne pourrait résoudre plusieurs problèmes. Du point de vue de la sécurité, le cloisonnement des applications dans une machine virtuelle permet de protéger des codes malveillants les ressources mises à disposition dans la grille. Inversement, le cloisonnement dans une machine virtuelle permet de protéger les codes des ressources malveillantes. Comme le montre [167], les machines virtuelles peuvent également constituer une solution pour sauvegarder l'état d'une application dans l'optique de le restaurer en cas de défaillance. Cela est alors simple à réaliser puisqu'il suffit de copier la machine virtuelle et de l'enregistrer pour la redémarrer sur un autre nœud lorsqu'une défaillance se produit. Finalement, les machines virtuelles permettent de simplifier la gestion de l'hétérogénéité de ressources. D'un point de vue matériel, il est en effet possible d'émuler n'importe quelle architecture sur les ressources afin de pouvoir exécuter des applications nécessitant de s'exécuter sur une ressource dont l'architecture est rare dans la grille. D'un point de vue logiciel, l'utilisation d'une machine virtuelle permet de simplifier le choix des ressources puisque le système d'exploitation et les librairies nécessaires à l'exécution peuvent être embarquées dans la machine virtuelle. Plusieurs travaux comme [137], [80] ou encore [82] s'intéressent à l'utilisation de machines virtuelles dans un contexte de grille.

Shell de grille

Pour soumettre une application à une grille dont les ressources sont fédérées par Vigne, les utilisateurs doivent décrire les besoins en termes de ressources matérielles et logicielles et les relations entre les différentes tâches qui composent l'application. Il serait intéressant d'exploiter le concept de SIU à l'extrême en concevant un *shell* de grille. Les utilisateurs pourraient lancer leurs applications aussi simplement qu'ils le feraient sur une station de travail.

Cette fonctionnalité fait partie des objectifs que le projet XtreemOS [36] souhaite atteindre. En particulier, le module AEM (*Application Execution Management*) propose de fournir deux interfaces pour la soumission des applications. Une interface similaire à celle d'un gestionnaire de traitement par lots pour les applications ayant des besoins particuliers et une interface de type *shell* pour les applications n'ayant pas de pré-requis particulier.

Intégration d'un modèle économique

Si l'on se place dans le cas d'application d'une grille dont les ressources sont partagées entre plusieurs entreprises et institutions, il serait intéressant d'intégrer un modèle économique à Vigne. Il faudrait pour cela intégrer la notion de prix de ressources et de contrat entre les utilisateurs et les fournisseurs de ressource au sein du système. Les services concernés par une telle extension sont le système d'information, le service d'allocation de ressources et le service de gestion d'application. Compte-tenu de leur architecture, ces services pourraient être modifiés assez simplement pour mettre en œuvre un ordonnancement suivant un modèle économique. De plus, l'architecture du système d'information pourrait très simplement supporter

un modèle où les prix sont ajustés selon une loi dépendante de l'offre et de la demande. En effet, une ressource recevant beaucoup de demandes d'allocation pourrait augmenter son prix et inversement.

Pour facturer aux utilisateurs les ressources consommées, le système doit proposer des mécanismes permettant de comptabiliser précisément les consommations pour chaque utilisateurs. Afin de garantir la non-répudiation des données comptabilisées, ce mécanisme doit être utilisé avec une infrastructure d'authentification sécurisée des utilisateurs.

Transfert de concepts dans XtreemOS

XtreemOS [36] est un projet ambitieux donc l'objectif est de construire un système d'exploitation qui étend le système Linux pour gérer les organisations virtuelles d'une grille. XtreemOS couvre un ensemble de fonctionnalités beaucoup plus vaste que Vigne en proposant par exemple un système de fichier distribué, des mécanismes d'authentification, un support avancé pour la tolérance aux défaillances ou encore une interface POSIX. Néanmoins, certains concepts de Vigne pourraient être transférés dans XtreemOS. En effet, XtreemOS pourrait profiter de nos propositions pour ce qui concerne le système d'information, l'allocation efficace de ressources pour les applications composées de tâches communicantes ou la gestion d'applications de type *workflow*.

Grille de services et applications persistantes

Nous terminons par la proposition d'une perspective à plus long terme. Certaines fédérations de ressources sont vouées à être utilisées dans le contexte d'un nombre limité de domaines d'application. Typiquement, l'étude du climat fait intervenir un nombre limité de physiques. Ainsi, des solveurs permettant de résoudre un type de problème du domaine seront utilisés par de nombreux utilisateurs.

Dans un tel contexte, notre proposition pourrait être étendue pour intégrer la notion de service telle qu'elle est proposée par l'OGF [23] ou par les projets comme DIET [64]. Une application exécutée sur la grille pourrait être conservée par le nœud utilisé pour son exécution afin d'être utilisée pour un autre utilisateur.

Un système d'information distribué tel que celui que nous avons proposé pourrait être étendu pour inclure la notion de recherche de service dans la grille.

Pour assurer le meilleur service aux utilisateurs, des mécanismes dynamiques de réplication des applications pourraient être envisagés pour ajuster la durée de persistance des applications sur les nœuds de la grille. Ainsi, des répliques pourraient être créées lorsque la demande augmente ou elles pourraient être supprimées lorsque la demande diminue. Les méthodes permettant de détecter la popularité d'une application et celles permettant de répartir des répliques sur la grille pourraient utiliser des techniques similaires à celles proposées dans cette thèse pour faire connaître des nœuds qui possèdent des ressources rares dans la grille.

Bibliographie

[1] « *Altair PBS Professionnal* ». http://www.pbsgridworks.com.

[2] « *ApGrid (Asia Pacific Grid)* ». http://www.apgrid.org.

[3] « *The Beowulf Cluster Site* ». http://www.beowulf.org.

[4] « *Berkeley Lab Checkpoint/Restart (BLCR)* ». http://ftg.lbl.gov/CheckpointRestart/CheckpointRestart.shtml.

[5] « *Code_Aster* ». http://www.code-aster.org.

[6] « *Code_Saturne* ». http://rd.edf.com/code_saturne.

[7] « *DAGMan* ». http://www.cs.wisc.edu/condor/dagman.

[8] « *Distributed Resource Management Application API Working Group (DRMAA-WG)* ». http://drmaa.org/wiki.

[9] « *Grid'5000* ». http://www.grid5000.org.

[10] « *Gridbus Workflow Engine (GWFE)* ». http://gridbus.csse.unimelb.edu.au/workflow.

[11] « *GridLab* ». http://www.gridlab.org.

[12] « *GridLab Mercury Monitor* ». http://www.gridlab.org/WorkPackages/wp-11.

[13] « *GridLab Resource Management System* ». http://www.gridlab.org/WorkPackages/wp-9/index.html.

[14] « *Hawkeye - A Monitoring and Management Tool for Distributed Systems* ». http://www.cs.wisc.edu/condor/hawkeye.

[15] « *IBM LoadLeveler* ». http://www.ibm.com/systems/clusters/software/loadleveler.html.

[16] « *InfiniBand* ». http://www.infinibandta.org.

[17] « *Java Remote Method Invocation - Distributed Computing for Java* ». http://java.sun.com/javase/technologies/core/basic/rmi/whitepaper/index.jsp.

[18] « *JXTA* ». http://www.sun.com/software/jxta.

[19] « *KaZaa* ». http://www.kazaa.com.

[20] « *Message Passing Interface Forum* ». http://www.mpi-forum.org.

[21] « *Myrinet* ». http://www.myri.com/myrinet/overview.

[22] « *Napster protocol specification* ». http://opennap.sourceforge.net/napster.txt.

[23] « *Open Grid Forum (OGF)* ». http://www.ogf.org.

[24] « *OpenMP* ». http://www.openmp.org.

[25] « *OpenSSI (Single System Image) Clusters for Linux* ». http://openssi.org.

[26] « *Padico: a Software Environment for Computational Grids* ». http://padico.gforge.inria.fr.

[27] « *PlanetLab* ». http://www.planet-lab.org.

[28] « *Platform LSF* ». http://www.platform.com.

[29] « *Quadrics* ». http://www.quadrics.com.

[30] « *RENATER (Réseau National de télécommunications pour la Technologie l'Enseignement et la Recherche)* ». http://www.renater.fr.

[31] « *SALOME : The Open Source Integration Platform for Numerical Simulation* ». http://www.salome-platform.org.

[32] « *Sun Grid Engine* ». http://www.sun.com/software/gridware.

[33] « *TeraGrid* ». http://www.teragrid.org.

[34] « *TOP500 Supercomputer Sites* ». http://www.top500.org.

[35] « *Torque Resource Manager* ». http://www.clusterresources.com/pages/products/torque-resource-manager.php.

[36] « *XtreemOS* ». http://www.xtreemos.eu.

[37] David ALDOUS and Jim FILL. *Reversible Markov Chains and Random Walks on Graphs*. Disponible sur http://www.stat.berkeley.edu/~aldous/RWG/book.html.

[38] R. ALFIERI, R. CECCHINI, V. CIASCHINI, L. DELL'AGNELLO, Á. FROHNE, A. GIANOLI, K. LÕRENTEY and F. SPATARO. « VOMS, an Authorization System for Virtual Organizations ». In *Proceedings of the First European Across Grids Conference*, volume 2970 of *Lecture Notes in Computer Science*, pages 33–40, Santiago de Compostela, Spain, February 2004. Springer.

[39] William ALLCOCK, John BRESNAHAN, Rajkumar KETTIMUTHU and Michael LINK. « The Globus Striped GridFTP Framework and Server ». In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing (SC'05)*, page 54, Washington, DC, USA, 2005. IEEE Computer Society.

[40] Iyad ALSHABANI, Richard OLEJNIK, Bernard TOURSEL, Marek TUDRUJ and Eryk LASKOWSKI. « A Framework for Desktop GRID Applications: CCADAJ ». In *Proceedings of the Fifth International Symposium on Parallel and Distributed Computing (ISPDC'06)*, volume 0, pages 208–214, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[41] David P. ANDERSON. « BOINC: A System for Public-Resource Computing and Storage ». In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 4–10. IEEE Computer Society, 2004.

[42] David P. ANDERSON, Jeff COBB, Eric KORPELA, Matt LEBOFSKY and Dan WERTHIMER. « SETI@home: an experiment in public-resource computing ». *Communications of the ACM*, 45(11):56–61, November 2002.

[43] Nazareno ANDRADE, Lauro COSTA, Guilherme GERMOGLIO and Walfredo CIRNE. « Peer-to-peer grid computing with the OurGrid Community ». In *23rd Brazilian Symposium on Computer Networks*, 2005.

[44] Stephen Elbert ANDREW CHIEN, Brad Calder and Karan BHATIA. « Entropia: architecture and performance of an enterprise desktop grid system ». *Journal of Parallel and Distributed Computing*, 63(5):597–610, May 2003.

[45] Gabriel ANTONIU, Luc BOUGÉ and Mathieu JAN. « JuxMem: An Adaptive Supportive Platform for Data Sharing on the Grid ». *Scalable Computing: Practice and Experience*, 6(3):45–55, September 2005.

[46] R. ARMSTRONG, D. GANNON, A. GEIST, K. KEAHEY, S. KOHN, L. MCINNES, S. PARKER and B. SMOLINSKI. « Toward a Common Component Architecture for High-Performance Scientific Computing ». In *Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-99)*. IEEE Computer Society, 1999.

[47] Laurent BADUEL, Françoise BAUDE, Denis CAROMEL, Arnaud CONTES, Fabrice HUET, Matthieu MOREL and Romain QUILICI. « *Grid Computing: Software Environments and Tools* », Chapitre Programming, Deploying, Composing, for the Grid. Springer-Verlag, January 2006.

[48] A. BARAK, A. SHILOH and L. AMAR. « An organizational grid of federated MOSIX clusters ». In *IEEE International Symposium on Cluster Computing and the Grid, 2005. CCGrid 2005*, pages 350–357, Cardiff, UK, May 2005.

[49] Amnon BARAK and Oren LA'ADAN. « The MOSIX multicomputer operating system for high performance cluster computing ». *Future Generation Computer Systems*, 13(4-5):361–372, 1998.

[50] Paul BARHAM, Boris DRAGOVIC, Keir FRASER, Steven HAND, Tim HARRIS, Alex HO, Rolf NEUGEBAUER, Ian PRATT and Andrew WARFIELD. « Xen and the art of virtualization ». In *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP '03)*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[51] Sujoy BASU, Sujata BANERJEE, Puneet SHARMA and Sung-Ju LEE. « NodeWiz: peer-to-peer resource discovery for grids ». In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid 2005 (CCGrid 2005)*, pages 213–220, Cardiff, UK, May 2005.

[52] Olivier BEAUMONT, Anne-Marie KERMARREC, Loris MARCHAL and Etienne RIVIÈRE. « VoroNet: a scalable object network based on Voronoi Tessellations ». In *Proceedings of 21st IEEE International Parallel & Distributed Processing Symposium (IPDPS 2007)*, pages 1–10, Long Beach (CA), USA, 2007.

[53] Rüdiger BERLICH, Marcus HARDT, Marcel KUNZE, Malcolm ATKINSON and David FERGUSSON. « EGEE: building a pan-European grid training organisation ». In *Proceedings of the 2006 Australasian workshops on Grid computing and e-research (ACSW Frontiers '06)*, pages 105–111, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[54] Rüdiger BERLICH, Marcel KUNZE and Kilian SCHWARZ. « Grid computing in Europe: from research to deployment ». In *Proceedings of the 2005 Australasian workshop on Grid computing and e-research (ACSW Frontiers '05)*, pages 21–27, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[55] Francine BERMAN, Richard WOLSKI, Henri CASANOVA, Walfredo CIRNE, Holly DAIL, Marcio FAERMAN, Silvia FIGUEIRA, Jim HAYES, Graziano OBERTELLI, Jennifer SCHOPF, Gary SHAO, Shava SMALLEN, Neil SPRING, Alan SU and Dmitrii ZAGORODNOV. « Adaptive Computing on the Grid Using AppLeS ». *IEEE Transactions on Parallel and Distributed Systems*, 14(4):369–382, 2003.

[56] Ashwin R. BHARAMBE, Mukesh AGRAWAL and Srinivasan SESHAN. « Mercury: supporting scalable multi-attribute range queries ». In *Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '04)*, pages 353–366, New York, NY, USA, 2004. ACM Press.

[57] Raphaël BOLZE, Franck CAPPELLO, Eddy CARON, Michel DAYDÉ , Frederic DESPREZ, Emmanuel JEANNOT, Yvon JÉGOU, Stéphane LANTERI, Julien LEDUC, Noredine MELAB, Guillaume MORNET, Raymond NAMYST, Pascale PRIMET, Benjamin QUETIER, Olivier RICHARD, El-Ghazali TALBI and Touché IRENA. « Grid'5000: a large scale and highly reconfigurable experimental Grid testbed ». *International Journal of High Performance Computing Applications*, 20(4):481–494, nov 2006.

[58] Aurelien BOUTEILLER, Thomas HERAULT, Geraud KRAWEZIK, Pierre LEMARINIER and Franck CAPPELLO. « MPICH-V Project: A Multiprotocol Automatic Fault Tolerant MPI ». *International Journal of High Performance Computing Applications*, 20(3):319–333, 2006.

[59] Ali Raza BUTT, Rongmei ZHANG and Y. Charlie HU. « A Self-Organizing Flock of Condors ». *Journal of Parallel and Distributed Computing*, 66(1):145–161, 2006.

[60] Rajkumar BUYYA, David ABRAMSON and Jonathan GIDDY. « Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid ». In *The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region*, volume 01, page 283, Beijing, China, May 2000. IEEE Computer Society.

[61] Rajkumar BUYYA and Srikumar VENUGOPAL. « The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report ». In *Proceedings of the First IEEE International Workshop on Grid Economics and Business Models (GECON 2004)*, Seoul, Korea, April 2004. IEEE Press, New Jersey, USA.

[62] Nicolas CAPIT, Georges Da COSTA, Yannis GEORGIOU, Guillaume HUARD, Cyrille MARTIN, Grégory MOUNIE, Pierre NEYRON and Olivier RICHARD. « A batch schedu-

ler with high level components ». In *IEEE International Symposium on Cluster Computing and the Grid, 2005. CCGrid 2005*, pages 776–783, Cardiff, UK, May 2005.

[63] Franck CAPPELLO, Samir DJILALI, Gilles FEDAK, Thomas HERAULT, Frédéric MAGNIETTE, Vincent NÉRI and Oleg LODYGENSKY. « Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid ». *Future Generation Computer Systems*, 21:417–437, 2005.

[64] Eddy CARON and Frédéric DESPREZ. « DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid ». *International Journal of High Performance Computing Applications*, 20(3):335–352, 2006.

[65] Miguel CASTRO, Manuel COSTA and Antony ROWSTRON. « Should we build Gnutella on a structured overlay ? ». *SIGCOMM Comput. Commun. Rev.*, 34(1):131–136, 2004.

[66] Steve J. CHAPIN, Dimitrios KATRAMATOS, John F. KARPOVICH and Andrew S. GRIMSHAW. « The Legion Resource Management System ». In *IPPS/SPDP '99/JSSPP '99: Proceedings of the Job Scheduling Strategies for Parallel Processing*, pages 162–178, London, UK, 1999. Springer-Verlag.

[67] Jeffrey S. CHASE, Darrell C. ANDERSON, Prachi N. THAKAR, Amin M. VAHDAT and Ronald P. DOYLE. « Managing energy and server resources in hosting centers ». In *Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP '01)*, pages 103–116, New York, NY, USA, 2001. ACM Press.

[68] Walfredo CIRNE, Daniel PARANHOS, Lauro COSTA, Elizeu SANTOS-NETO, Francisco BRASILEIRO, Jacques SAUVE, Fabricio A. B. SILVA, Carla O. BARROS and Cirano SILVEIRA. « Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach ». In *International Conference on Parallel Processing (ICPP'03)*, pages 407–416. IEEE Computer Society, 2003.

[69] Georges Da COSTA, Corine MARCHAND, Olivier RICHARD and Jean-Marc VINCENT. « Resources availability for Peer to Peer systems ». In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA'06)*, volume 1, pages 21–28, Washington, DC, USA, 2006. IEEE Computer Society.

[70] Karl CZAJKOWSKI, Ian FOSTER and Carl KESSELMAN. « Resource Co-Allocation in Computational Grids ». In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, pages 219–228. IEEE Computer Society, 1999.

[71] Alexandre DENIS, Christian PÉREZ and Thierry PRIOL. « PadicoTM: An Open Integration Framework for Communication Middleware and Runtimes ». *Future Generation Computer Systems*, 19(4):575–585, May 2003.

[72] Samir DJILALI. « P2P-RPC: Programming Scientific Applications on Peer-to-Peer Systems with Remote Procedure Call ». In *Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid (CCGRID'03)*, page 406, Tokyo, Japan, May 2003. IEEE Computer Society.

[73] Niels DROST, Rob V. van NIEUWPOORT and Henri E. BAL. « Simple locality-aware co-allocation in peer-to-peer supercomputing ». In *Proceedings of the Sixth Interna-*

tional Workshop on Global and Peer-2-Peer Computing (GP2P), volume 2, page 14, Singapore, May 2006.

[74] D. EASTLAKE and P. JONES. « US Secure Hash Algorithm 1 (SHA1) ». RFC 3174, Internet Engineering Task Force, 2001.

[75] P. EEROLA, B. KÓNYA, O. SMIRNOVA, T. EKELÖF, M. ELLERT, J. R. HANSEN, J. L. NIELSEN, A. WÄÄNÄNEN, A. KONSTANTINOV, J. HERRALA, M. TUISKU, T. MYKLEBUST, F. OULD-SAADA and B. VINTER. « The NorduGrid production Grid infrastructure, status and plans ». In *Proceedings of the Fourth International Workshop on Grid Computing (GRID '03)*, pages 158–165, 2003.

[76] D.H.J. EPEMA, M. LIVNY, R. van DANTZIG, X. EVERS and J. PRUYNE. « A worldwide flock of Condors: Load sharing among workstation clusters ». *Future Generation Computer Systems*, 12:53–65, 1996.

[77] P. ERDÕS and S. J. TAYLOR. « Some intersection properties of random walk paths ». *Acta Mathematica Hungarica*, 11(3–4):231–248, September 1960.

[78] Dror G. FEITELSON and Larry RUDOLPH. « Gang Scheduling Performance Benefits for Fine-Grained Synchronization ». *Journal of Parallel and Distributed Computing*, 16(4):306–318, December 1992.

[79] Matthieu FERTRÉ and Christine MORIN. « Extending a Cluster SSI OS for Transparently Checkpointing Message-Passing Parallel Applications ». In *Proceedings of the International Symposium on Parallel Architectures Algorithms, and Networks (I-SPAN05)*, Las Vegas, Nevada, USA, December 2005.

[80] Renato J. FIGUEIREDO, Peter A. DINDA and José A. B. FORTES. « A Case For Grid Computing On Virtual Machines ». In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS '03)*, page 550, Washington, DC, USA, 2003. IEEE Computer Society.

[81] Organization for the Advancement of STRUCTURED INFORMATION STANDARDS (OASIS). « UDDI Version 3.0.2 ». Specification Technical Committee Draft 3.0.2, Organization for the Advancement of Structured Information Standards, October 2004.

[82] Ian FOSTER, Thomas FREEMAN, Kate KEAHY, Douglas SCHEFTNER, Borja SOTOMAYER and Xuehai ZHANG. « Virtual Clusters for Grid Communities ». In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, pages 513–520, Washington, DC, USA, 2006. IEEE Computer Society.

[83] Ian FOSTER and Adriana IAMNITCHI. « On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing ». In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, February 2003.

[84] Ian FOSTER and Carl KESSELMAN. « Globus: A Metacomputing Infrastructure Toolkit ». *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[85] Ian FOSTER and Carl KESSELMAN. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, an Francisco, CA, USA, 1st edition, July 1998.

[86] Ian FOSTER, Carl KESSELMAN and Steve TUECKE. « The Anatomy of the Grid: Enabling Scalable Virtual Organizations ». *International Journal of Supercomputing Applications*, 15(3), 2002.

[87] OpenLDAP FOUNDATION. « Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map ». RFC 4510, Internet Engineering Task Force, June 2006.

[88] Eitan FRACHTENBERG, Dror G. FEITELSON, Fabrizio PETRINI and Juan FERNANDEZ. « Adaptive Parallel Job Scheduling with Flexible Coscheduling ». *IEEE Transactions on Parallel and Distributed Systems*, 16(11):1066–1077, 2005.

[89] R. FREUND, T. KIDD, D. HENSGEN and L. MOORE. « SmartNet: a scheduling framework for heterogeneous computing ». In *Proceedings of the International Symposium on Parallel Architectures*, page 514. IEEE Computer Society, 1996.

[90] James FREY, Todd TANNENBAUM, Miron LIVNY, Ian FOSTER and Steven TUECKE. « Condor-G: A Computation Management Agent for Multi-Institutional Grids ». *Cluster Computing*, 5(3):237–246, July 2002.

[91] Pascal GALLARD and Christine MORIN. « Dynamic Streams For Efficient Communications between Migrating Processes in a Cluster ». *Parallel Processing Letters*, 13(4):601–614, December 2003.

[92] Ayalvadi J. GANESH, Anne-Marie KERMARREC and Laurent MASSOULIÉ. « Peer-to-Peer membership management for gossip-based protocols ». *IEEE Transactions on Computers*, 52(2):139–149, February 2003.

[93] Yolanda GIL, Pedro A. GONZÁLEZ-CALERO and Ewa DEELMAN. « On the black art of designing computational workflows ». In *Proceedings of the 2nd workshop on Workflows in support of large-scale science (WORKS '07)*, pages 53–62, Monterey Bay, California, USA, June 2007. In conjunction with HPDC 2007.

[94] Christos GKANTSIDIS, Milena MIHAIL and Amin SABERI. « Random Walks in Peer-to-Peer Networks ». In *Proceedings of the Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004)*, page 130, Hong Kong, March 2004.

[95] Andrzej GOSCINSKI, Michael HOBBS and Jack SILCOCK. « GENESIS: an efficient, transparent and easy to use cluster operating system ». *Parallel Computing*, 28(4):557–606, April 2002.

[96] Krishna P. GUMMADI, Richard J. DUNN, Stefan SAROIU, Steven D. GRIBBLE, Henry M. LEVY and John ZAHORJAN. « Measurement, modeling, and analysis of a peer-to-peer file-sharing workload ». In *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP'03)*, pages 314–329, New York, NY, USA, 2003. ACM Press.

[97] Fabien HANTZ and Hervé GUYENNET. « HiPoP: Highly Distributed Platform of Computing ». In *Proceedings of the IEEE joint International Conference on Autonomic and Autonomous Systems (ICAS'05) and International Conference on Networking and Services (ICNS'05)*, pages 91–96, Tahiti, French Polynesia, October 2005.

[98] Fabien HANTZ and Hervé GUYENNET. « A P2P Platform using sandboxing ». In *Proceedings of the Workshop on Security and High Performance Computing Systems (HPCS'06), in conjunction with 20th European Conference on Modelling and Simulation (ECMS'06)*, pages 736–739, Bonn, Germany, May 2006.

[99] Adriana IAMNITCHI and Ian T. FOSTER. « On Fully Decentralized Resource Discovery in Grid Environments ». In *Proceedings of the Second International Workshop on Grid Computing (Grid'01)*, pages 51–62, London, UK, 2001. Springer-Verlag.

[100] Emmanuel JEANVOINE. « Distributed operating system for resource discovery and allocation in federated clusters ». Poster presented at the twentieth ACM symposium on Operating systems principles (SOSP '05), October 2005.

[101] Emmanuel JEANVOINE, Christine MORIN and Daniel LEPRINCE. « Un protocole de découverte de ressources optimisé pour l'allocation de ressources dans les grilles ». In *Actes de la 5ème Conférence Française en Systèmes d'Exploitations (CFSE-5 2006)*, pages 49–59, Perpignan, France, October 2006. ACM-SIGOPS de France.

[102] Emmanuel JEANVOINE, Christine MORIN and Daniel LEPRINCE. « Vigne: Executing Easily and Efficiently a Wide Range of Distributed Applications in Grids ». In *Proceedings of Euro-Par 2007*, volume 4641 of *Lecture Notes in Computer Science*, pages 384–393, Rennes, France, August 2007. Springer.

[103] Emmanuel JEANVOINE, Louis RILLING, Christine MORIN and Daniel LEPRINCE. « Architecture distribuée pour la gestion des ressources dans des grilles à grande échelle ». In *Actes de la 6ème Conférence Internationale sur les Nouvelles Technologies de la Répartition (NOTERE 2006)*, pages 231–242, Toulouse, France, June 2006. Hermes.

[104] Emmanuel JEANVOINE, Louis RILLING, Christine MORIN and Daniel LEPRINCE. « Using Overlay Networks to Build Operating System Services for Large Scale Grids ». *Scalable Computing: Practice and Experience*, 8(3):229–239, 2007.

[105] Yvon JEGOU. « Dynamic Memory Management on Mome DSM ». In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, page 16, Washington, DC, USA, 2006. IEEE Computer Society.

[106] Berit JOHANNES. « Scheduling parallel jobs to minimize the makespan ». *Journal of Scheduling*, 9(5):433–452, 2006.

[107] Binu K. JOHNSON, R. KARTHIKEYAN and D. Janaki RAM. « DP: A Paradigm for Anonymous Remote Computation and Communication for Cluster Computing ». *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1052–1065, 2001.

[108] Nirav H. KAPADIA and José A. B. FORTES. « PUNCH: An architecture for Web-enabled wide-area network-computing ». *Cluster Computing*, 2(2):153–164, 1999.

[109] Nicholas T. KARONIS, Brian TOONEN and Ian FOSTER. « MPICH-G2: A Grid-enabled implementation of the Message Passing Interface ». *Journal of Parallel and Distributed Computing*, 63:551–563, 2003.

[110] Patrick KIRK. « Gnutella Protocol Development ». `http://www.the-gdf.org`.

[111] Sébastien LACOUR. « *Contribution à l'automatisation du déploiement d'applications sur des grilles de calcul* ». Thèse de doctorat, Université de Rennes 1, IRISA, Rennes, France, December 2005.

[112] Barry LAWSON and Evgenia SMIRNI. « Power-aware resource allocation in high-end systems via online simulation ». In *Proceedings of the 19th annual international conference on Supercomputing (ICS'05)*, pages 229–238, New York, NY, USA, 2005. ACM Press.

[113] Tobin J. LEHMAN and James H. KAUFMAN. « OptimalGrid: Middleware for Automatic Deployment of Distributed FEM Problems on an Internet-Based Computing Grid ». In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'03)*, page 164. IEEE Computer Society, 2003.

[114] Mike LEWIS and Andrew GRIMSHAW. « The Core Legion Object Model ». In *Fifth IEEE International Symposium on High Performance Distributed Computing 1996*, Syracuse, NY, USA, 1996.

[115] Hui LI, David GROEP, Lex WOLTERS and Jeff TEMPLON. « Job Failure Analysis and Its Implications in a Large-Scale Production Grid ». In *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing (E-SCIENCE'06)*, page 27, Washington, DC, USA, 2006. IEEE Computer Society.

[116] Michael LITZKOW, Miron LIVNY and Matthew MUTKA. « Condor - A Hunter of Idle Workstations ». In *8th International Conference of Distributed Computing Systems*, June 1988.

[117] Oleg LODYGENSKY, Gilles FEDAK, Franck CAPPELLO, Vincent NERI, Miron LIVNY and Doug THAIN. « XtremWeb & Condor sharing resources between Internet connected Condor pools. ». In *Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid (CCGRID'03)*, page 382, Tokyo, Japan, May 2003. IEEE Computer Society.

[118] Matthew L. MASSIE, Brent N. CHUN and David E. CULLER. « The ganglia distributed monitoring system: design, implementation, and experience ». *Parallel Computing*, 30(7):817–840, July 2004.

[119] Laurent MASSOULIÉ, Erwan Le MERRER, Anne-Marie KERMARREC and Ayalvadi GANESH. « Peer counting and sampling in overlay networks: random walk methods ». In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing (PODC'06)*, pages 123–132, New York, NY, USA, 2006. ACM Press.

[120] Carlo MASTROIANNI, Domenico TALIA and Oreste VERTA. « A super-peer model for resource discovery services in large-scale Grids ». *Future Generation Computer Systems*, 21:1235–1248, 2005.

[121] Raissa MEDEIROS, Walfredo CIRNE, Francisco BRASILEIRO and Jacques SAUVÉ. « Faults in grids: why are they so bad and what can be done about it? ». In *Proceedings of Fourth International Workshop on Grid Computing (Grid'2003)*, pages 18–24, Washington, DC, USA, 2003. IEEE Computer Society.

[122] Andrey MIRTCHOVSKI, Rob SIMMONDS and Ron MINNICH. « Plan 9 - An Integrated Approach to Grid Computing ». In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop 17*, page 273a, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[123] H.H. MOHAMED and D.H.J. EPEMA. « The Design and Implementation of the KOALA Co-Allocating Grid Scheduler ». In *Proceedings of the European Grid Conference*, volume 3470 of *Lecture Notes in Computer Science*, pages 640–650. Springer-Verlag, 2005.

[124] Christine MORIN, Pascal GALLARD, Renaud LOTTIAUX and Geoffroy VALLÉE. « Towards an Efficient Single System Image Cluster Operating System ». *Future Generation Computer Systems*, 20(2):505–521, January 2004.

[125] Matt W. MUTKA and Miron LIVNY. « Profiling Workstations' Available Capacity for Remote Execution ». In *Proceedings of the 12th IFIP WG 7.3 International Symposium on Computer Performance Modelling, Measurement and Evaluation (Performance '87)*, pages 529–544, Amsterdam, The Netherlands, 1988. North-Holland Publishing Co.

[126] Harvey B. NEWMAN, Iosif C. LEGRAND, Philippe GALVEZ, R. VOICU and C. CIRSTOIU. « MonALISA: a distributed monitoring service architecture ». In *Proceedings of the Conference on Computing in High Energy and Nuclear Physics (CHEP 2003)*, La Jolla, CA, USA, March 2003.

[127] Object Management Group (OMG). « CORBA Component Model, v3.0 ». Technical Report formal/2002-06-65, Object Management Group, June 2002.

[128] Object Management Group (OMG). « CORBA/IIOP Specification v3.0.3 ». Technical Report formal/2004-03-01, Object Management Group, March 2004.

[129] David OPPENHEIMER, Jeannie ALBRECHT, David PATTERSON and Amin VAHDAT. « Design and implementation tradeoffs for wide-area resource discovery ». In *Proceedings of 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14 2005)*, pages 113–124, 2005.

[130] Shrideep PALLICKARA and Geoffrey FOX. « NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids ». In *Proceedings of ACM/IFIP/USENIX International Middleware Conference (Middleware-2003)*, volume 2672 of *Lecture Notes in Computer Science*, pages 41–61, Rio de Janeiro, Brazil, June 2003. Springer.

[131] J. N. PAN. « Reliability prediction of imperfect switching systems subject to Weibull failures ». *Computers and Industrial Engineering*, 34(2):481–492, 1998.

[132] Patricia PASCAL. « *Gestion de ressources pour des services déportés sur des grappes d'ordinateurs avec qualité de service garantie* ». Thèse de doctorat, INSA, LAAS, Toulouse, France, November 2004.

[133] Laura PEARLMAN, Von WELCH, Ian FOSTER, Carl KESSELMAN and Steven TUECKE. « A Community Authorization Service for Group Collaboration ». In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, page 50, 2002.

[134] Rob PIKE, Dave PRESOTTO, Sean DORWARD, Bob FLANDRENA, Ken THOMPSON and Phil TRICKEY, Howard Winterbottom. « Plan 9 from Bell Labs ». *Computing Systems*, 8(3):221–254, 1995.

[135] Diego PUPPIN, Stefano MONCELLI, Ranieri BARAGLIA, Nicola TONELLOTTO and Fabrizio SILVESTRI. « A Grid Information Service Based on Peer-to-Peer ». In José C. CUNHA and Pedro D. MEDEIROS, editors, *Proceedings of Euro-Par 2005*, Lisbon, Portugal, August 2005. Published as Lecture Notes in Computer Science Volume 3648 / 2005.

[136] Christian PÉREZ, Thierry PRIOL and André RIBES. « A Parallel CORBA Component Model for Numerical Code Coupling ». *The International Journal of High Performance Computing Applications*, 17(4):417–429, 2003.

[137] Lavanya RAMAKRISHNAN, David IRWIN, Laura GRIT, Aydan YUMEREFENDI, Adriana IAMNITCHI and Jeffrey S. CHASE. « Toward a doctrine of containment: grid hosting with adaptive resource control ». In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing (SC '06)*, page 101, Tampa, Florida, USA, 2006. ACM Press.

[138] Rajesh RAMAN, Miron LIVNY and Marvin SOLOMON. « Matchmaking: Distributed Resource Management for High Throughput Computing ». In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC 7)*, Chicago, IL, July 1998.

[139] Rajesh RAMAN, Miron LIVNY and Marvin SOLOMON. « Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching ». In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12 '03)*, pages 80–89. IEEE Computer Society, 2003.

[140] Sylvia RATNASAMY, Paul FRANCIS, Mark HANDLEY, Richard KARP and Scott SCHENKER. « A scalable content-addressable network ». In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM'01)*, pages 161–172. ACM Press, 2001.

[141] Sean RHEA, Dennis GEELS, Timothy ROSCOE and John KUBIATOWICZ. « Handling Churn in a DHT ». In *Proceedings of the 2004 USENIX Technical Conference*, pages 127–140, Boston, MA, USA, June 2004.

[142] André RIBES and Christian CAREMOLI. « Salomé platform component model for numerical simulation ». In *Proceedings of the Second IEEE International Workshop on Software Architectures and Component Technologies (SACT 07)*, pages 553–564, Bejing, China, July 2007.

[143] Louis RILLING. « *Système d'exploitation à image unique pour une grille de composition dynamique : conception et mise en oeuvre de services fiables pour exécuter les applications distribuées partageant des données* ». Thèse de doctorat, Université de Rennes 1, IRISA, Rennes, France, November 2005.

[144] Louis RILLING. « Vigne: Towards a Self-Healing Grid Operating System ». In *Proceedings of Euro-Par 2006*, volume 4128 of *Lecture Notes in Computer Science*, pages 437–447, Dresden, Germany, August 2006. Springer.

[145] Louis RILLING and Christine MORIN. « A Practical Transparent Data Sharing Service for the Grid ». In *Proceedings of the Fifth International Workshop on Distributed Shared Memory (DSM 2005)*, pages 897–904, Cardiff, UK, May 2005.

[146] R. RIVEST. « The MD5 Message-Digest Algorithm ». RFC 1321, Internet Engineering Task Force, 1992.

[147] Mathilde ROMBERG. « The UNICORE Architecture: Seamless Access to Distributed Resources ». In *Proceedings of the the Eighth International Symposium on High Performance Distributed Computing*, pages 287–293. IEEE Computer Society, August 1999.

[148] Thomas ROPARS. « Supervision d'applications sur grille de calcul ». Rapport de master de recherche, Université de Rennes 1, June 2006.

[149] Thomas ROPARS, Emmanuel JEANVOINE and Christine MORIN. « GAMoSe: An Accurate Monitoring Service for Grid Applications ». In *Proceedings of the 6th International Symposium on Parallel and Distributed Computing (ISPDC 2007)*, page 40, Hagenberg, Austria, July 2007. IEEE Computer Society.

[150] Antony I. T. ROWSTRON and Peter DRUSCHEL. « Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems ». In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, 2001. Springer-Verlag.

[151] Dolors ROYO, Luis Diaz de CERIO, Nirav H. KAPADIA and Jose A. B. FORTES. « Active Yellow Pages: A Pipelined Resource Management Architecture for Wide-Area Network Computing ». In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10 '01)*, page 0147. IEEE Computer Society, 2001.

[152] Kyung Dong RYU and Jeffrey K. HOLLINGSWORTH. « Unobtrusiveness and Efficiency in Idle Cycle Stealing for PC Grids ». In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, page 62a, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[153] Stefan SAROIU, Krishna P. GUMMADI and Steven D. GRIBBLE. « Measuring and analyzing the characteristics of Napster and Gnutella hosts ». *Multimedia Systems*, 9(2):170–184, 2003.

[154] Albert SEMTNER. « Ocean and climate modeling ». *Communication of the ACM*, 43(4):80–89, April 2000.

[155] Edi SHMUELI and Dror G. FEITELSON. « Backfilling with lookahead to optimize the packing of parallel jobs ». *Journal of Parallel and Distributed Computing*, 65(9):1090–1107, 2005.

[156] K. SHUDO, Y. TANAKA and S. SEKIGUCHI. « P3: P2P-based middleware enabling transfer and aggregation of computational resources ». In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid*, volume 1, pages 259–266, Cardiff, UK, May 2005.

[157] Ion STOICA, Robert MORRIS, David KARGER, M. Frans KAASHOEK and Hari BALAKRISHNAN. « Chord: A scalable peer-to-peer lookup service for internet applications ». In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM'01)*, pages 149–160. ACM Press, 2001.

[158] Daniel STUTZBACH and Reza REJAIE. « Understanding churn in peer-to-peer networks ». In *Proceedings of the 6th ACM SIGCOMM on Internet measurement (IMC'06)*, pages 189–202, New York, NY, USA, 2006. ACM Press.

[159] Kumaran SUBRAMONIAM, Muthucumaru MAHESWARAN and Michel TOULOUSE. « Towards a micro-economic model for resource allocation in Grid computing systems ».

In *Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering (CCECE'02)*, volume 2, pages 782–785, 2002.

[160] V. S. SUNDERAM. « PVM: a framework for parallel distributed computing ». *Concurrency: Practice and Experience*, 2(4):315–339, 1990.

[161] Domenico TALIA and Paolo TRUNFIO. « Toward a Synergy between P2P and Grids ». *IEEE Internet Computing*, 7(4):94–96, 2003.

[162] Yoshio TANAKA, Hidemoto NAKADA, Satoshi SEKIGUCHI, Toyotaro SUZUMURA and Satoshi MATSUOKA. « Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing ». *Journal of Grid Computing*, 1(1):41–51, 2003.

[163] Osamu TATEBE, Noriyuki SODA, Youhei MORITA, Satoshi MATSUOKA and Satoshi SEKIGUCHI. « Gfarm v2: A Grid file system that supports high-performance distributed and parallel data computing ». In *Proceedings of the 2004 Computing in High Energy and Nuclear Physics Conference (CHEP'04)*, Interlaken, Switzerland, September 2004.

[164] Ian TAYLOR, Ian WANG, Matthew SHIELDS and Shalil MAJITHIA. « Distributed computing with Triana on the Grid: Research Articles ». *Concurrency and Computation: Practice & Experience*, 17(9):1197–1214, 2005.

[165] Jang uk IN, Paul AVERY, Richard CAVANAUGH, Laukik CHITNIS, Mandar KULKARNI and Sanjay RANKA. « SPHINX: A Fault-Tolerant System for Scheduling in Dynamic Grid Environments ». In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, page 12b, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[166] Gladys UTRERA, Julita CORBALÁN and Jesús LABARTA. « Another approach to backfilled jobs: applying virtual malleability to expired windows ». In *Proceedings of the 19th annual international conference on Supercomputing (ICS'05)*, pages 313–322, New York, NY, USA, 2005. ACM Press.

[167] Geoffroy VALLEE, Thomas NAUGHTON, Hong ONG and Stephen SCOTT. « Checkpoint/Restart of Virtual Machines Based on Xen ». In *Proceedings of the High Availability and Performace Computing Workshop (HAPCW 2006)*, Santa Fe, New Mexico, USA, October 2006.

[168] Gopinath Tarun VENKATESWARA REDDY M., Vijay Srinivas A. and Janakiram D.. « Vishwa: A Reconfigurable Peer-to-Peer Middleware for Grid Computations ». In *Proceedings of the 35th International Conference on Parallel Processing*, pages 381–390, Ohio, USA, August 2006. IEEE Computer Society.

[169] Von WELCH, Frank SIEBENLIST, Ian Foster John BRESNAHAN, Karl CZAJKOWSKI, Jarek GAWOR, Carl KESSELMAN, Sam MEDER, Laura PEARLMAN and Steven TUECKE. « Security for Grid Services ». In *Proceedings of the Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, June 2003.

[170] Brian WHITE, Michael WALKER, Marty HUMPHREY and Andrew S. GRIMSHAW. « LegionFS: A Secure and Scalable File System Supporting Cross-Domain High-Performance Applications ». In *Proceedings of the ACM/IEEE Supercomputing Conference (SC'01)*, page 59, Denver, Colorado, USA, nov 2001. ACM/IEEE, ACM Press.

[171] J. WROCLAWSKI. « The Use of RSVP with IETF Integrated Services ». RFC 2210, Internet Engineering Task Force, 1997.

[172] Asim YARKHAN, Keith SEYMOUR, Kiran SAGI, Zhiao SHI and Jack DONGARRA. « Recent Developments in GridSolve ». *International Journal of High Performance Computing Applications*, 20(1):131–141, 2006. Special Issue: Scheduling for Large-Scale Heterogeneous Platforms.

[173] Chee Shin YEO and Rajkumar BUYYA. « Pricing for Utility-driven Resource Management and Allocation in Clusters ». In *Proceedings of the 12th International Conference on Advanced Computing and Communication (ADCOM 2004)*, pages 32–41, Ahmedabad, India, 2004. Allied Publishers: New Delhi, India.

[174] Yingwu ZHU, Xiaoyu YANG and Yiming HU. « Making Search Efficient on Gnutella-Like P2P Systems ». In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, page 56a. IEEE Computer Society, 2005.

[175] George Kingsley ZIPF. « Selected Studies of the Principle of Relative Frequency in Language ». *Language (Harvard University Press)*, 9(1):89–92, March 1933.

Résumé

La simulation numérique, de plus en plus utilisée dans de nombreux domaines scientifiques, requiert une grande puissance de calcul. L'émergence des grilles de calcul permet d'envisager l'accès à une puissance de calcul, jusqu'à présent inégalée, qui est tout à fait adaptée aux simulations numériques. Toutefois, les caractéristiques d'une grille telles que la grande échelle, la volatilité de ses noeuds ou encore leur hétérogénéité rendent son utilisation complexe.

Cette thèse étudie la conception de services de niveau système pour l'exécution d'applications distribuées dans des grilles de très grande taille. Notre approche s'articule autour de trois axes.

Tout d'abord, nous avons conçu des services complètement distribués pour éliminer les risques de goulet d'étranglement et de point critique sensible aux défaillances. Ces services sont fondés sur l'utilisation de réseaux pair-à-pair qui possèdent des propriétés d'auto-réparation et d'auto-organisation. En particulier nous avons conçu un système d'information reposant sur un réseau logique non-structuré et sur une optimisation du protocole de marches aléatoires. Ensuite, nous proposons une conception suffisamment générique pour que nos services puissent être utilisés avec n'importe quel type de grille et une large gamme d'applications distribuées. Finalement, l'ensemble des services que nous proposons concourent à l'exécution simple, efficace et fiable des applications. Pour simplifier l'utilisation d'une grille, notre approche consiste à fournir une vision de type système à image unique qui permet de masquer la distribution des ressources d'une grille aux utilisateurs. Nous proposons un service d'allocation de ressources permettant de profiter au mieux des ressources de la grille en tenant compte des besoins particuliers des applications comme les communications entre tâches. Nous proposons aussi un service de gestion d'application doté de mécanismes de supervision de noeuds et de tâches qui garantit que l'exécution d'une applications puisse atteindre son terme en dépit des défaillances et des reconfigurations intervenant dans la grille.

Nos propositions ont été mises en oeuvre dans le prototype Vigne conçu au sein de l'équipe-projet PARIS de l'INRIA. Des évaluations de ce prototype par simulation et sur la plate-forme Grid'5000 avec un grand nombre de noeuds ont permis de montrer la faisabilité et l'efficacité de nos propositions.

Finalement, nous avons couplé le système Vigne à la plate-forme de simulation numérique SALOME. Dans ce contexte, nous avons validé notre approche avec une application réelle d'envergure de type *workflow* fournie par EDF R&D.

Mots clés

Intergiciel, grille de calcul, système à image unique, exécution fiable, découverte de ressources, allocation de ressources, gestion d'application.