LaPIe : Communications Collectives adaptées aux Grilles de Calcul

Luiz Angelo Barchet-Estefanel

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

Thèse

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : "Informatique : Systèmes et Communications"

préparée au laboratoire INFORMATIQUE ET DISTRIBUTION
dans le cadre de ***l'École Doctorale "Mathématiques, Sciences et Technologies de l'Information, Informatique"***

présentée et soutenue publiquement par

Luiz Angelo BARCHET-ESTEFANEL

le 2 décembre 2005

LaPIe : Communications Collectives adaptées aux Grilles de Calcul

---

Directeur de thèse : Mr Denis Trystram
Co-Directeur de thèse : Mr Grégory Mounié

JURY

MR JEAN-CLAUDE KÖNIG,	Président
MR OLIVIER BEAUMONT,	Rapporteur
MR THILO KIELMANN,	Rapporteur
MR ALFREDO GOLDMAN,	Examinateur extérieur
MR DENIS TRYSTRAM,	Examinateur (directeur de thèse)
MR GRÉGORY MOUNIÉ,	Examinateur (co-encadrant)

I walk the maze of moments
But everywhere I turn to
Begins a new beginning
But never finds a finish
I walk to the horizon
And there I find another
It all seems so surprising
And then I find that I know
Anywhere Is, Enya

À Manuele

Remerciements

L'élaboration de cette thèse n'a pas été possible sans le support et l'amitié de plusieurs personnes. J'aimerais tout d'abord remercier M. Denis Trystram et M. Grégory Mounié, mes directeur et co-directeur de thèse, qui m'ont accueilli dans leur équipe et m'ont supporté tout au long de ce travail. Les conseils toujours sages et sincères de Denis et la curiosité inépuisable de Grégory ont marqué indélébilement l'élaboration de ce travail.

Je dois aussi remercier très fortement les membres de mon jury : M. Jean-Claude König, président du jury, M. Thilo Kielmann et M. Olivier Beaumont, les rapporteurs et M. Alfredo Goldman, examinateur. Leur lecture attentive et la pertinence de leurs remarques ont permis l'amélioration de la qualité de ce document.

J'aimerais aussi remercier la CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pour m'avoir soutenu financièrement pendant le développement de ma thèse.

Je tiens à remercier les amis Céline Lopez, Yan Grunenberger et Maxime Martinasso, ainsi qu'aux collègues brésiliens Leonardo Brenner et Mauricio Pillon. Je remercie également tous les autres, mais la liste serait évidemment trop longue... L'important est que votre amitié m'ait ouvert des nouveaux horizons.

Je remercie aussi mes parents et mon frère, ainsi que ma belle-famille, pour leur support inconditionnel malgré la distance.

À ma femme, Manuele, à qui je dois la plupart de cette thèse. C'est son rêve de venir en France qui m'a contagié, et qui nous a permis de trouver le « chez nous » dans un pays si lointain. Pendant ces années de travail, lorsque j'étais confus et découragé, elle me donnait la force nécessaire ; lorsque je suivais les mauvaises pistes, elle me rappelait à mes objectifs. Pendant toutes ces années elle a été (et sera toujours) la personne la plus importante de ma vie ! Merci énormément, mon amour.

Table des matières

Table des figures

Liste des algorithmes

Liste des tableaux

1

Introduction

1.1 Contexte général

Depuis quelques années le domaine du calcul à haute performance a beaucoup évolué. Les supercalculateurs, ou machines parallèles à mémoire distribuée, ont fait place à des systèmes plus vastes, composés de machines standard. Ces ressources ont d'abord été regroupées en grappes de calcul, qui contenaient soit des machines déjà présentes dans le réseau local (les *Networks of Workstations* - NOWs), soit des machines spécifiquement acquises pour faire partie de ces grappes de calcul.

C'est à partir des années 90, grâce à la dissémination de l'Internet grand public, que l'idée de rendre l'accès aux ressources de calcul plus généralisé a pris forme. Dans [FOS99], Ian Foster et Karl Kesselman comparent ce partage des ressources de calcul avec l'accès à la puissance électrique délivrée par les fournisseurs. C'est alors que l'interconnexion de ressources géographiquement distantes et potentiellement hétérogènes devient la prochaine étape de l'évolution des systèmes parallèles. En effet, l'évolution des technologies d'interconnexion, notamment en ce qui concerne le débit des liaisons de longue distance (comme le réseau français à Vraiment Très Haut Débit - VTHD[1]), a permis l'interconnexion et le partage des ressources de calcul de plusieurs partenaires scientifiques. Ceci a donné naissance à ce qu'on appelle les *grilles de calcul* ou *metacomputing*, selon le point de vue matériel ou logiciel.

Malheureusement, ce type de système est bien plus complexe au sens où l'hétérogénéité augmente le nombre de paramètres à maîtriser. Leur utilisation est parfois très limitée par manque d'outils adéquats et performants, en particulier pour ce qui concerne les communications entre les processus parallèles. En effet, une bonne partie des applications lancées sur les grilles de calcul sont de type multiparamétrique, dont chaque instance effectue des calculs avec peu ou aucune communication vers les autres processus.

Dans le cas des paradigmes d'exécution où les processus parallèles communiquent entre eux, la performance des communications devient un facteur essentiel. C'est le cas des communications collectives, opérations qui correspondent aux patrons de communication à plusieurs interlocuteurs, et qui sont des outils puissants et indispensables pour la computation parallèle. Normalement, les implantations des communications collectives (par exemple dans la bibliothèque MPI) sont adaptées aux environnements de type réseau local et ne sont pas optimisées pour les environnements fortement hétérogènes comme les grilles de calcul. Le travail effectué dans le cadre de cette thèse vise alors

[1] http ://www.vthd.org

l'étude du comportement des communications collectives et leur modélisation en vue de concevoir des algorithmes performants adaptés aux grilles de calcul.

1.2 Problématique

Plusieurs travaux récents visent l'implantation des opérations de communication collective adaptées aux systèmes à grande échelle, notamment les grilles de calcul. Dans ces environnements, l'hétérogénéité est un facteur prépondérant qui doit obligatoirement être pris en compte. Cette hétérogénéité représente néanmoins un vrai défi pour la prédiction des performances, car les facteurs qui influencent les communications ont des origines très variées, comme la distribution des processus (par exemple, sur une grappe de machines multiprocesseurs), la distance entre les machines et/ou les grappes, le taux d'utilisation du matériel (surtout la congestion du réseau) et la variation de performance du matériel. En effet, très souvent les grilles de calcul combinent différentes machines et réseaux.

L'hétérogénéité intrinsèque à ces environnements, associée à la constante évolution des ressources disponibles sur les grilles de calcul, empêche la création d'opérations spécifiques pour ces environnements, comme en attestent [BHA03] et [VAD00]. Pour simplifier cette modélisation, la plupart des solutions considèrent les grilles comme l'interconnexion d'îlots de grappes homogènes. Dans ce contexte, la majorité des systèmes concentre l'optimisation au niveau des communications entre les grappes, puisque ces liaisons sont généralement plus lentes que celles intérieures à la grappe. Quelques exemples de cette approche en deux couches incluent les bibliothèques ECO [LOW96], MagPIe [KIE99b][KIE01] et même la bibliothèque LAM-MPI 7 [LAM04], laquelle considère les machines SMP comme des îlots de communication rapide.

Il existe toutefois la possibilité d'organiser les communications en un plus grand nombre de couches. En effet, le travail de Karonis et *al.* [KAR00][KAR02] a démontré que le découpage en plusieurs couches de communication peut conduire à des réductions du temps d'exécution plus importantes qu'un découpage en deux couches et, pour cela, il est nécessaire d'avoir la connaissance *a priori* du coût de communication interne à chaque grappe. Dans ce cas, le calcul de la distribution et de la hiérarchie des communications dépend des temps de communication à l'intérieur des grappes, qui varient selon l'opération de communication collective, le nombre de noeuds et les caractéristiques du réseau de chaque grappe.

Dans ce travail, nous adressons le problème de l'optimisation des opérations collectives sur des grilles de calcul à travers deux aspects fondamentaux et complémentaires, la modélisation des performances et l'étude de techniques d'ordonnancement des communications.

1.3 Contributions

Les contributions apportées par ce travail s'insèrent dans plusieurs domaines liés aux systèmes parallèles et repartis. Comme principales contributions, nous pouvons citer :

La découverte de topologie

La découverte de topologie a un rôle très significatif dans notre travail, car son utilité réside notamment dans l'identification d'hétérogénéités sur une grille de calcul. Cette identification est très importante dans le contexte des optimisations proposées dans ce travail, une fois que la présence d'hétérogénéités augmente la complexité des modèles et réduit la précision des prédictions de performance.

Cependant, l'environnement d'optimisation mis en place dans ce travail requiert des propriétés qui ne sont pas facilement respectées par les outils traditionnels de découverte de topologie, tels que NWS [WOL97]. C'est ainsi que nous développons notre propre méthodologie de découverte

de topologie, qui mélange à la fois des techniques traditionnelles et des techniques de mesure de performance. Cette méthodologie, adaptée à l'environnement des applications parallèles, a pour objectif la découverte des caractéristiques du réseau dans toute son extension, avec une analyse très pragmatique des aspects qualitatives et quantitatives des interconnexions réseau.

La modélisation des performances

Nos contributions dans le domaine de la modélisation de performance portent sur le développement et la validation de modèles de coût pour les opérations de communication collective. À partir de trois patrons de communication représentatifs, *Un vers Plusieurs*, *Un vers Plusieurs Personnalisé* et *Plusieurs vers Plusieurs Personnalisé*, nous étudions les différents aspects qui influencent les opérations de communication collective.

Dans le cas des patrons *Un vers Plusieurs* et *Un vers Plusieurs Personnalisé*, notre apport s'est concentré sur l'étude des différentes stratégies de communication et le développement de modèles de performance capables de représenter le fonctionnement de ces stratégies. Ceci représente une relecture et une extension des modèles existants dans la littérature, comme par exemple les stratégies proposées par Huse [HUS99] et Vadhiyar [VAD00, VAD04].

Dans le cas du patron de communication *Plusieurs vers Plusieurs Personnalisé* nous avons constaté que les modèles couramment cités dans la littérature ne sont pas capables de représenter les conditions réelles de congestion du réseau. Cette observation nous a poussé vers l'étude de nouveaux modèles de performance, mieux adaptés aux situations réelles. Pour cela, nous augmentons les modèles de Christara [CHR99] et Pjesivac-Grbovic [PJE05], de manière à rendre plus précise et efficace la prédiction de performances, sans pour autant augmenter la complexité du modèle, comme dans le cas du modèle de coût LoGPC [MOR01].

L'optimisation des communications collectives pour les grilles de calcul

Dans le domaine de l'optimisation des communications collectives pour les grilles de calcul, nos contributions principales portent sur la validation de techniques adaptées à cet environnement. Parmi ces techniques, nous étudions notamment les heuristiques d'ordonnancement de communication de type hiérarchique. En effet, nous travaillons sur l'hypothèse que les communications collectives sur les grilles de calcul peuvent être organisées en plusieurs couches hiérarchiques dynamiquement organisées. Cette hypothèse représente une évolution par rapport aux implantations classiques pour les grilles comme ECO [LOW96], MagPIe [KIE99b, KIE00] ou le travail de Karonis *et al.* [KAR00], où les communications sur grille suivent un ordre fixe pré-établi.

À partir de l'hypothèse en plusieurs couches, nous cherchons des heuristiques efficaces pour l'ordonnancement des communications. À l'aide de simulations et expériences pratiques, nous comparons le fonctionnement des différentes heuristiques, à la recherche de stratégies d'optimisation de communication à la fois efficaces et robustes. Cette analyse nous permet notamment l'identification des facteurs qui affectent l'ordonnancement des communications et qui permettent le développement de nouvelles heuristiques, plus adaptées aux besoins des futures générations de grilles de calcul.

1.4 Organisation du manuscrit

Ce document est divisé en trois parties principales. Une première partie présente les aspects liés à la découverte de topologie, outil essentiel pour réduire la complexité des techniques d'optimisation étudiées dans le reste du document. Ainsi, dans le chapitre 2, nous rappelons les principales techniques de découverte de topologie, avec une analyse de leurs caractéristiques et de nos besoins. Cette analyse nous conduit à la définition d'un *framework* pour la découverte de topologie adaptée aux nécessités des applications de calcul parallèle, comme décrit dans le chapitre 3. Toutefois, comme la découverte de topologie n'est pas exclusivement liée au calcul parallèle, nous présentons en Annexe

A un exemple d'application de la découverte de topologie en vue d'optimiser la performance d'un protocole de Diffusion Totalement Ordonnée, opération caractéristique des systèmes répartis.

La deuxième partie permet d'introduire la problématique de la modélisation des performances des communications collectives dans un environnement homogène de type grappe d'ordinateurs. Pour ce faire, dans le chapitre 4 nous rappelons les principaux modèles de performance existants en vue d'identifier les aspects nécessaires à la modélisation des communications collectives. Nous présentons ensuite, dans le chapitre 5, une révision des patrons de communication collective et des opérations MPI qui correspondent à chaque patron. Dernier chapitre de cette partie, le chapitre 6 présente nos expériences dans le domaine de la modélisation des communications collectives, notamment avec la validation des modèles à travers des expériences pratiques.

Finalement, la troisième partie est consacrée à l'étude des techniques d'optimisation des communications collectives pour les environnements de type grille de calcul. Le chapitre 7 présente une étude sur les techniques d'optimisation des communications collectives, notamment le développement d'heuristiques d'ordonnancement adaptées à l'organisation hiérarchique des noeuds dans un tel environnement.

I

Découverte de Topologie

2

Découverte de Topologie

La connaissance de la topologie du réseau est un atout indispensable pour la distribution efficace des applications, et pour l'optimisation des communications entre les processus parallèles. Une utilisation de la connaissance de la topologie est le placement des processus parallèles, selon leur nombre et leurs besoins en ressources de calcul [PAK03]. Cependant, quand les ressources nécessaires ne sont plus contraintes à une seule grappe, le simple placement des processus ne suffit pas pour garantir la performance de l'application, spécialement si les applications échangent intensivement des messages.

Afin d'assurer une performance de communication compatible avec le nombre de ressources employées, nous nous intéressons à la découverte de topologie comme un outil pour augmenter la connaissance du réseau, afin de faciliter la construction des primitives de communication collective adaptées aux environnements de type grille de calcul. Puisque les environnements de grille sont intrinsèquement hétérogènes et comptent un nombre important de composants, l'optimisation des communications collectives ne permet plus l'utilisation de stratégies exhaustives comme celles proposées par Bhat [BHA03] ou Vadhiyar[VAD00]. En effet, ces propositions considèrent soit la génération d'arbres de communication à partir des données d'interconnexion de tous les noeuds, soit l'expérimentation pratique pour déterminer les meilleurs paramètres d'utilisation.

Une solution adoptée par plusieurs auteurs pour réduire la complexité des optimisations est la sous-division du réseau en différentes couches de communication. Ainsi, l'approche la plus simple est d'établir une séparation entre les communications de courte et longue distance, ce qui permet, par exemple, l'optimisation des communications entre deux grappes distantes, normalement plus lentes que les communications à l'intérieur des grappes. Un autre avantage de cette stratégie est que les ressources à l'intérieur d'une grappe sont normalement homogènes, ce qui facilite la modélisation des communications et par conséquent, réduit la complexité des optimisations. Quelques exemples de cette approche en « deux couches » sont les systèmes ECO [LOW96, LOW00] et MagPIe [KIE99b], mais nous pouvons aussi considérer l'implantation pour machines SMP de la bibliothèque LAM-MPI 7 [LAM04]. D'un autre côté, le découpage en plus de deux couches n'est pas une pratique courante, malgré les résultats présentés par Karonis [KAR00, KAR02] qui démontrent la possibilité d'augmenter la performance des communications collectives par l'utilisation de plusieurs couches de communications, adaptées à la topologie du réseau et au coût des communications à l'intérieur des grappes.

L'approche suivie par ce travail est celle des multiples couches de communication. En effet,

nous utilisons des outils de découverte de topologie pour identifier au milieu de la grappe des « îlots d'homogénéité » ou *grappes logiques*. Associée à l'obtention de paramètres précis d'interconnexion du réseau, la découverte de topologie permet alors le découpage virtuel du réseau et la réorganisation des couches de communication pour augmenter l'efficacité des communications collectives.

2.1 Caractéristiques recherchées

Nous nous intéressons à la découverte automatique de la topologie effective du réseau, en particulier les temps de communications entre les noeuds au niveau applicatif, et non à une vue précise et détaillée du réseau tel que le schéma de câblage physique ou le matériel de routage. Notre but est de permettre le regroupement des noeuds connectivement proches en îlots d'homogénéité (des grappes logiques). Dû à la diversité des patrons de communication, les opérations de communication collective ne sont pas nécessairement structurées comme des arbres avec une racine unique ; dans ce cas-là la connaissance des performances du réseau de bout en bout (*end-to-end*) entre tous les noeuds est nécessaire. À cela s'ajoute le besoin de connaissance des performances au niveau de l'application, notamment entre processus MPI distribués sur les noeuds de la grille.

Par conséquent, la découverte de topologie dans le cadre de notre travail doit satisfaire les trois contraintes suivantes :

Précision au niveau de l'application. L'environnement de l'application comporte normalement des services qui peuvent influencer la connectivité entre les noeuds. L'outil de découverte de topologie doit s'adapter à de tels environnements, et délivrer des informations adaptées à l'application.

Minimisation du nombre de mesures. Dû au grand nombre de noeuds qui composent une grille, la réalisation des tests de bout-en-bout entre tous les noeuds s'avère une tâche longue et intrusive. Afin de réduire les perturbations induites sur la grille, il faut limiter le nombre de tests. Par exemple, il suffit de mesurer la connectivité entre une paire de machines pour déduire celle de toutes les machines (similaires) sur un réseau local.

Exhaustivité. Pour toutes paires de machines, si aucun test direct n'est réalisé entre elles, le système doit pouvoir estimer leur connectivité par agrégation d'autres tests.

2.2 Découverte de topologie et la structure du réseau

Le modèle de réseau OSI (*Open Systems Interconnection* - interconnexion de systèmes ouverts) est constitué de sept couches, (cf. Figure 2.1). Chaque couche offre une vision différente du réseau, donc le niveau considéré pour la découverte du réseau influe à la fois la topologie, la façon de la découvrir et les informations recueillies. Généralement, les niveaux 2 et 3 sont les plus couramment utilisés pour la découverte de topologie. La couche 2 est nommée « liaison de données », et correspond aux protocoles spécifiques de chaque environnement réseau, par exemple, le protocole Ethernet. Par contre, la couche 3, nommée « réseau », correspond aux protocoles tels que le protocole internet (Internet Protocol - IP). La couche 2 est plus proche des liens physiques que la couche réseau et peut être utilisée pour obtenir des informations à propos des routeurs non disponibles depuis le niveau 3. En revanche, la couche 3 est plus proche de la vision que les applications ont du réseau.

Il existent toutefois des situations où, selon les besoins de l'application, il faut regarder les couches supérieures pour mieux représenter la connectivité entre les noeuds et/ou les processus. Ainsi, par exemple, certaines grilles de calcul utilisent des mécanismes tels que les VLAN (*Virtual LAN*), qui font partie des couches supérieures des protocoles réseau. Cette technologie permet aux administrateurs de regrouper comme un seul réseau plusieurs liens distincts, comme dans le cas du projet GRID5000 [GRI05]. De la même façon, des *proxys* de type IMPI [IMPI] peuvent être utilisés

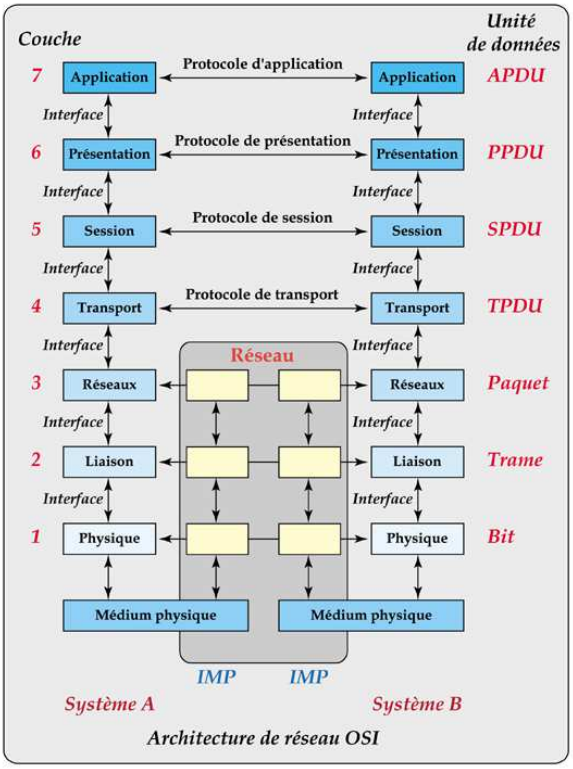


Figure 2.1 *Le modèle en couches OSI*

pour interconnecter différentes implantations de MPI ou des sous-réseaux séparés par des pare-feux. Il est donc nécessaire de tenir compte de tels mécanismes lors de l'établissement de méthodes de cartographie automatique du réseau, et la façon la plus fiable est d'utiliser directement les couches supérieures du modèle OSI pour obtenir la topologie effective du réseau.

Dans les prochaines sections nous présentons les approches les plus utilisées pour la découverte de topologie, suivies d'une analyse de leurs avantages et désavantages par rapport à nos objectifs.

2.2.1 Topologie fournie par l'utilisateur

Certains outils et bibliothèques de communication adaptés aux environnements de grille (*grid-aware*) organisent leurs communications à partir des informations de topologie fournies par l'utilisateur, comme dans le cas de MagPIe [KIE99b] et MPICH-G2 [KAR03].

Pourtant, malgré sa simplicité, les topologies fondées sur la localité des machines ne sont pas suffisantes pour garantir la précision des modèles de communication et en particulier la construction de communications à plusieurs niveaux. Plus exactement, des paramètres d'évaluation de type « sous-réseau IP » ne peuvent pas représenter la topologie effective du réseau, et par conséquent, peuvent cacher des hétérogénéités qui ne sont alors pas prises en compte par les modèles de communication.

Si les informations sur la topologie ne sont pas adaptées à la modélisation des communications collectives, cela n'exclut pas l'utilisation de données de localisation pour orienter la découverte de la topologie, qui doit néanmoins extraire des informations qui représentent les interconnexions effectives du réseau ou de la grille.

2.2.2 Protocoles de gestion du réseau - SNMP

La couche 2 du modèle OSI est celle qui définit les réseaux locaux (LAN). Il serait donc possible de consulter directement les composants du réseau en utilisant le protocole SNMP (*Simple Network Management Protocol* - protocole simple de gestion du réseau [BIE00]). Ainsi, plusieurs travaux comme ceux de Bejerano [BEJ03] et Breitbart [BRE00] utilisent SNMP pour construire la topologie des réseaux.

Malheureusement, pour que cette approche soit efficace, il faut que les administrateurs configurent correctement les dispositifs réseau et permettent l'accès à ces informations. En effet, il y a certains routeurs qui ne répondent pas à des requêtes SNMP pour raison de sécurité, tandis que d'autres demandent l'utilisation d'outils propriétaires fournis par leurs constructeurs tels que le protocole de découverte Cisco ou celui de Bay Networks.

Pour consulter les informations sur les réseaux longue distance (WAN), il est possible d'utiliser BGP (*Border Gateway Protocol* - protocole pour passerelles [REK95]), utilisé pour échanger des informations sur le routage entre les systèmes autonomes. En effet, le projet Remos [DIN01, LOW00] utilise cette approche, en déduisant la topologie du réseau local grâce à SNMP et celle du réseau à grande distance grâce à des tests bout à bout.

L'approche SNMP+BGP permet donc l'obtention de la configuration du réseau directement par des composants du réseau, ce que lui rend relativement rapide et cause peu de perturbations sur le réseau. Malheureusement, l'usage de ces protocoles est le plus souvent restreint à un petit nombre d'utilisateurs privilégiés. La raison principale est la sécurité, puisqu'il est possible de mener des attaques de type « déni de service ». De plus, les fournisseurs d'accès à Internet sont généralement réticents à permettre l'usage de SNMP sur leur réseau car cela reviendrait à publier les possibles défaillances de leur infrastructure, ce qui peut nuire à leur image commerciale. Effectivement, il est rare d'obtenir le droit d'utiliser SNMP sur les réseaux d'organisations dont on ne fait pas partie. Même si les plates-formes classiques de grilles impliquent le plus souvent plusieurs organisations bien établies telles que des universités, obtenir l'autorisation d'effectuer des opérations non classiques telles que l'accès aux protocoles de la couche liaison peut devenir très coûteuse en temps et en efforts en raison de facteurs humains.

2.2.3 Méthodes tomographiques

La tomographie est une méthode utilisée par exemple en imagerie médicale et consistant à reconstruire une vision tridimensionnelle d'un objet à partir de plusieurs vues bidimensionnelles. De manière comparable, différentes solutions existent pour reconstruire la topologie du réseau à partir d'informations collectées depuis différentes machines. Ces méthodes diffèrent principalement par la façon de collecter les visions locales de chaque machine.

ping

Le programme *ping* est classiquement utilisé pour mesurer le temps d'aller-retour (*Round Trip Time* - RTT) d'un paquet entre deux hôtes sur le réseau. Des projets tels que IDMaps [FRA01] ou Global Network Positioning (positionnement global sur le réseau, [NG01]) utilisent cet outil pour cartographier le réseau par partitionnement sur cette métrique de distance, c'est-à-dire en regroupant les machines semblant placées à la même distance de certains serveurs donnés. Cette approche malheureusement ne donne pas assez d'informations dans notre contexte puisque le traitement des paquets ICMP diffère beaucoup des paquets TCP utilisés par MPI, et les paramètres du réseau ne sont pas assez fiables pour être utilisés avec les modèles de performance.

traceroute

La couche 3 du modèle OSI est celle où est rendue possible l'interconnexion de différents réseaux. Pour éviter les boucles infinies, tous les paquets ont une durée de vie (*Time To Live* - TTL) déterminée lors de leur création, et enregistrée dans l'entête IP du paquet. Cette valeur est décrémentée par chaque routeur transmettant le paquet d'un réseau à un autre. Lorsqu'elle devient nulle, le paquet est détruit et un message d'erreur ICMP est envoyé à son émetteur. Comme la plupart des routeurs indiquent leur adresse dans le message d'erreur produit quand le TTL devient nul, *traceroute* peut découvrir tous les sauts nécessaires pour atteindre un hôte distant donné en émettant plusieurs paquets de TTL croissants. Cette fonctionnalité est utilisée par l'outil *traceroute*, mais aussi par le travail de Raz [RAZ02], qui présente des optimisations pour la découverte des routes de type *traceroute* en utilisant différents algorithmes de retransmission des paquets d'erreur.

L'outil *traceroute* est largement utilisé par certains projets de cartographie automatique tels que TopoMon [BUR02] et GIAS [LI01], que l'utilisent pour déterminer les interactions de flux de données à travers l'analyse des routes partagées entre différents noeuds. Néanmoins, les informations de connectivité obtenues par *traceroute* ne sont pas suffisamment précises, car les réponses, envoyées sous forme de paquets ICMP, dépendent beaucoup de la réactivité des routeurs et machines.

NWS

Le Network Weather Service [WOL97] (Service météorologique du réseau NWS) est un système distribué basé sur des senseurs logiciels permettant de regrouper des informations sur l'état actuel du réseau et des machines d'une plate-forme. Il est ainsi possible d'obtenir le débit et la latence de chaque lien TCP reliant deux hôtes abritant des senseurs NWS. De même, la charge processeur, les espaces mémoire et disque disponibles ou le nombre de processeurs de chaque hôte peuvent être mesurés.

NWS peut mesurer les performances en termes de bande passante et latence pour tout lien TCP/IP entre deux de ses senseurs. La latence est définie par le temps d'aller-retour (*Round-Trip Time*) d'un paquet de très petite taille : la machine souhaitant mesurer la latence chronomètre le temps nécessaire pour que la machine cible lui renvoie le paquet de quatre octets qu'elle lui a envoyé, selon Figure 2.2.

Le débit, par contre, est estimé en chronométrant le temps nécessaire à l'envoi des paquets de plus grande taille (par exemple 64 Ko) et en retirant le temps relatif à la latence de transmission des messages. Le choix de la valeur des paquets de grande taille permet de mesurer un réseau local classique tout en limitant l'intrusivité des expérimentations.

Il reste toutefois la contrainte que les mesures de NWS sont limitées par l'utilisation d'un seul *socket* sur une seule machine. En effet, cette approche n'est pas adaptée aux réseaux de très haut débit comme le VTHD (2,5Gb/s). Par conséquence, NWS détecte une bande passante maximale de l'ordre de 100 Mb/s sur VTHD [QUI03], même en réglant convenablement les paramètres des tests. Ceci indique clairement que le senseur NWS a atteint le maximum de bande passante qu'une application utilisant un seul *socket* réussit à occuper.

Il est aussi important de s'assurer qu'un même lien n'est jamais utilisé par deux tests de bande passante concurrents au même instant. Dans ce cas de collision entre deux mesures, les flux se partageraient la bande passante disponible sur le lien, ce qui fausserait les résultats (chaque test pourrait donner un résultat égal à la moitié de la valeur réelle). Pour éviter ce problème, NWS introduit la notion de clique : tous les hôtes appartenant à une même clique sont réputés partager une ressource réseau, et NWS s'assure que les tests menés entre ces liens n'entrent pas en collision. Un algorithme d'élection de leader par passage de jeton est utilisé pour cela, et seul l'hôte en possession du jeton à un instant donné est autorisé à initier un test réseau.

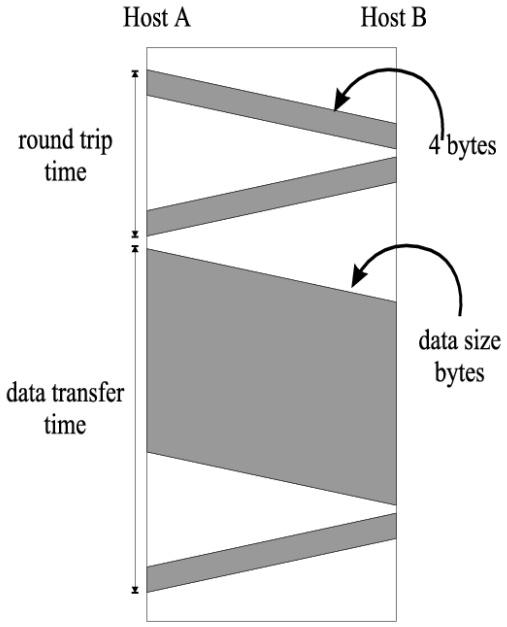


Figure 2.2 *Procédure de NWS pour l'obtention de la latence et du débit*

2.2.4 Détection d'interférences

TopoMon

Malgré la possibilité d'éviter la collision entre deux mesures par l'utilisation des cliques, NWS n'est pas capable d'assurer que les différentes cliques ne partagent pas des liens en commun. La détection de possibles interférences est alors une nécessité, d'autant plus que le routage WAN n'est pas facilement connu.

Dans le but d'identifier les liens partagés par différents noeuds, et de cette manière automatiser le déploiement de NWS, l'outil TopoMon [BUR02] utilise *traceroute*. Par évaluation des routes, TopoMon détecte les machines groupées sous le même réseau, et peut ainsi fournir à NWS des informations plus précises pour la disposition des cliques.

ENV - Effective Network View

Le projet Effective Network View (ENV [SHA99]) a pour principe de mesurer directement les interférences entre les flux de données. Pour cela, la bande passante normale entre deux machines données est comparée à celle obtenue lorsqu'un autre lien (entre deux autres machines) est saturé. Une variation indique que les deux liens partagent des ressources réseau, et cette information est utilisée pour construire la topologie d'interconnexion des machines. ENV utilise uniquement des expérimentations de niveau applicatif, ne nécessitant donc pas de privilèges ou d'outils spéciaux. En revanche, l'obtention de la vue complète de la topologie requiert une grande quantité d'expériences. En effet, il faut tester les éventuelles interactions du lien connectant chaque paire de machines avec chaque autre lien, ce qui conduit à un algorithme en $O(n^4)$ pour n machines. De plus, chaque expérience requiert que le réseau soit stabilisé, ce que ne permettrait de faire que quelques expériences par minute. L'approche choisie pour ENV alors est de représenter le réseau seulement du point

de vue d'une machine donnée, approche suffisante pour les applications qui suivent le paradigme maître/esclave.

ALNeM - Application-Level Network Mapper

L'une des restrictions de ENV est qu'il ne permet pas d'obtenir la topologie complète de la plate-forme, mais seulement une vue arborescente enracinée en un point arbitraire du réseau. Il est donc impossible d'obtenir par ce biais des informations sur les liens connectant des machines (autres que le maître choisi) entre elles. Malheureusement, certaines solutions de cartographie du réseau présentées précédemment ne sont pas adaptées à un contexte de *metacomputing*. Le projet ALNeM (Application-Level Network Mapper - outil de découverte de la topologie de niveau applicatif) [LEG03, QUI03, LEG04] vise à l'identification des constituants de la grille et leur distribution logique, obtenue à partir de leurs interactions réseau.

Afin d'accélérer le processus de découverte du réseau, ALNeM mène les expériences en parallèle en tirant parti de l'existence de liens indépendants. Puisqu'ils n'interfèrent pas les uns sur les autres, il est possible de saturer plusieurs de ces liens en parallèle, et une enquête plus approfondie est menée seulement si une baisse de performance est détectée lors des expérimentations de multiples liens en parallèle. Afin de prédire les liens probablement indépendants et guider l'algorithme de découverte du réseau, une étape préliminaire fait une approximation de la topologie grâce à *traceroute*.

2.3 Discussion

Les composants de la grille reposent le plus souvent sur une configuration manuelle pour obtenir ces informations, ce qui est une source d'erreur importante. Afin d'automatiser la cartographie de la plate-forme, il est possible d'obtenir les informations directement de la configuration telle qu'exprimée par les administrateurs grâce aux protocoles SNMP et BGP, mais cela requiert des autorisations spécifiques (en raison de problèmes de sécurité et de confidentialité), ce qui s'avérer problématique dans un contexte de *metacomputing*. Des outils tels que *ping* ou *traceroute* permettent l'obtention de paramètres de connexion, mais ces informations sont trop imprécises pour l'usage prévu, et parfois nécessitent même des privilèges spécifiques sur la machine utilisée pour confectionner les paquets de test, interdisant son usage dans le cadre du metacomputing.

Une exception est l'outil NWS. NWS est le leader incontesté de la surveillance de la grille dans la communauté, et présent des métriques de haut niveau alliées à la capacité de prédire les évolutions futures. Son approche, très réussie, s'adapte aussi aux environnements de large échelle grâce à la sous-division du réseau en cliques, partitions logiques qu'évitent la surcharge du réseau par des paquets de test.

Plus récemment, les auteurs se sont intéressés à l'étude des interactions entre les flux de données. Dans le cas précis de notre travail, ce sujet n'est pas un objectif prioritaire. Premièrement, les optimisations des opérations de communication collective tiennent déjà en compte la réduction du trafic sur les liens distants, qui représentent la plupart des liens dits « partagés ». Deuxièmement, notre stratégie de découverte du réseau utilise des techniques de *clustering* (partitionnement), c'est à dire, le regroupement des noeuds « proches », ce qui est équivalent à la détection des noeuds qui partagent un lien distant dans leur communication avec les autres noeuds.

C'est à partir du modèle NWS que nous proposons un *framework* spécifique pour la découverte de topologie adaptée aux besoins des applications parallèles de type MPI, dont nous cherchons à modéliser et optimiser les communications collectives. Le prochain chapitre détaille les étapes de notre modèle de découverte de topologie.

3

Découverte de Topologie pour les Applications MPI

Dans le chapitre précédent, nous avons étudié quelques outils pour la découverte de topologie. En ce qui concerne la découverte de topologie pour les applications parallèles, nous considérons que le découpage du réseau doit se faire préférentiellement par rapport à l'environnement de l'application, car l'environnement d'exécution peut influencer les temps de communication. Les outils de surveillance du réseau présentés précédemment, malgré leur importance, n'ont pas pour objectif l'obtention de paramètres de communication précis pour la modélisation des communications, et donc ne peuvent pas assurer la précision nécessaire à notre travail.

En effet, la plupart de ces outils acquièrent des informations à des niveaux assez proches de la couche matérielle, qui parfois ne reflète pas l'environnement de l'application. D'autres outils, comme ALNeM [LEG03, QUI03], sont plus intéressés aux interactions des liens de communication, qui malgré son importance, ont pour désavantage le coût excessif des mesures et une forte intrusivité sur le réseau. De surcroît, les informations de connectivité varient d'un outil à l'autre, alors que nous aimerions des informations de connectivité qui puissent être directement utilisées pour nos modèles de performance.

Une autre préoccupation de notre travail est de limiter l'intrusivité des procédures de découverte de topologie. En effet, même si quelques outils ont déjà des stratégies pour limiter le nombre de messages échangés et la perturbation sur le réseau, la propriété de **l'Exhaustivité** exigée par notre travail et présentée en page 26 n'est pas toujours respectée.

Dans ce travail, nous proposons une alternative mixte, adaptée à nos exigences pour la modélisation des performances de communication, mais avec une forte conscience sur la perturbation du réseau et la scalabilité des systèmes. Notre proposition, structurée comme un *framework* [BAR04b] dédié à la découverte automatique de la topologie du réseau, est utilisée dans l'objectif d'optimiser la modélisation des communications collectives et permettre la construction de primitives adaptées à l'environnement hétérogène des grilles de calcul.

Ainsi, nous proposons une approche mixte pour la découverte de topologie, qui se déroule en deux étapes : la première étape est responsable de la collecte des informations de connectivité des différents réseaux et de façon indépendante ; la deuxième étape, plus proche de l'application, sert à identifier la distribution des processus sur les machines, découper les réseaux en îlots de communi-

cation homogène (grappes logiques) et finalement obtenir les paramètres pLogP nécessaires à nos modèles de communication.

Comme la première partie est indépendante de l'application, elle peut utiliser des informations fournies par d'autres outils de surveillance du réseau (par exemple, NWS [WOL97]) voir des informations fournies pour l'utilisateur ou le service de réservation des noeuds (PBS [PBS], OAR [CAP03], etc.). Ces informations sont utilisées pour construire une matrice de distance qui servira de base à notre découverte de topologie. Pour une mesure de simplicité, nous considérons que cette matrice de distances est creuse, dans le sens que une grappe n'est pas obligée à surveiller ses interconnexions avec les autres grappes. Cette matrice peut contenir des différents paramètres de connectivité qui seront utilisés pour classifier les noeuds et les liens, notamment la latence et le débit.

La matrice de distance obtenue dans la première partie est alors communiquée à la deuxième partie de notre outil, normalement exécutée au début de l'application (inséré dans le code de *MPI_Init*). Les processus sont alors organisés en sous-réseaux homogènes, et les paramètres d'interconnexion entre ceux sous-réseaux sont acquis. Parce que le réseau est structuré en sous-réseaux homogènes, l'obtention des paramètres se fait d'une manière optimisée, réduisant ainsi l'impact des mesures sur la performance du réseau et l'initialisation de l'application.

De cette manière, à la fin du processus de découverte de topologie nous avons des grappes logiques composées de machines homogènes, et des paramètres d'interconnexion suffisamment précis pour permettre la construction des arbres d'interconnection qui optimisent à la fois les communications collectives à l'intérieur et à l'extérieur des grappes.

3.1 Première partie : collecte des informations

La plupart des travaux sur l'optimisation des communications collectives pour les environnements de grille considèrent, par simplicité, que les grappes sont définies par sa localisation physique ou le sous-réseau IP, et que toutes les machines à l'intérieur d'une grappe sont similaires. Or, cette hypothèse de localité n'est pas totalement adaptée aux systèmes réels, qui peuvent contenir des machines avec différentes performances de calcul et de communication. En effet, même les grappes avec des machines identiques présentent des performances différentes (nous croyons qu'il est pratiquement impossible d'avoir l'homogénéité de ressources dans une grappe avec quelques centaines de machines). Par conséquent, le choix de la topologie du réseau doit préférablement se faire par des aspects opérationnels, qui reflètent la performance réelle des machines et des liens.

Comme expliqué dans le chapitre précédent, l'acquisition des informations de connectivité peut se faire à travers des outils de surveillance du réseau. Ces outils peuvent obtenir des informations à partir des tests directs, comme par exemple *traceroute* ou NWS [WOL97], à partir de requêtes SNMP comme REMOS [LOW00], ou même en associant les deux approches, comme l'outil TopoMon [BUR02]. Par simplicité, nous considérons que ces informations sont obtenues au niveau de l'utilisateur, avec des tests similaires à ceux de NWS. Nous avons choisi NWS par le fait qu'il est l'outil *de facto* de la communauté grille, et qu'il peut fournir des diverses informations comme la latence, le débit, l'utilisation des CPUs et la mémoire disponible sur les machines. Nous sommes particulièrement intéressés pour les informations de latence et débit, plus utiles pour identifier les groupes de machines avec de paramètres de communication similaires.

Une différence important de notre méthode, par rapport à d'autres outils comme TopoMon, est que à cette étape nous n'avons pas besoin des interconnexions entre tous les noeuds des grappes. En effet, l'objectif de cette première partie est de découvrir les hétérogénéités cachées à l'intérieur des grappes, et pour cela, chaque grappe peut utiliser son propre système de surveillance, sans avoir besoin de contacter les autres grappes. Ainsi, cette stratégie permet la réduction du nombre de messages échangés sur le réseau, mais aussi permet l'utilisation des informations des services de surveillance, généralement plus corrects que des observations ponctuelles.

Les données de connectivité des différentes grappes sont réunies dans une unique matrice de distance. Comme les grappes ne sont pas conscientes des autres grappes, les interconnexions « manquantes » clairement délimitent les frontières de chaque grappe, réduisant ainsi le coût du processus de partitionnement qui sera effectué dans la deuxième partie de notre découverte de topologie.

3.2 Deuxième partie : partitionnement

À partir des données d'interconnexion obtenues dans la partie précédente, notre objectif initial est de regrouper les noeuds en grappes logiques différentes. Pour cela, des algorithmes de partitionnement (*clustering*) se font nécessaires.

Le partitionnement des noeuds peut se faire à travers l'organisation hiérarchique des noeuds, ou alors à travers le simple découpage en groupes isolés (bulles). Un exemple de la première approche est présenté par [JAI91], et le résultat est une arborescence où les noeuds proches sont regroupés sous les mêmes branches de l'arbre. Le désavantage de cette méthode est qu'elle présente une structure figée, qui n'est pas nécessairement adaptée à l'ensemble de communications collectives existantes.

L'autre approche, au contraire, se limite à regrouper les noeuds qui ont une différence ρ préalablement définie. Pour cela, on considère un graphe $G = (V, E)$, composé de $V = \{1, 2, \ldots, |V|\}$ noeuds et des arcs $\{i, j\}$, chacun avec un poids $E_{j,i}$. La matrice de distances M est définie comme

$$M = \begin{cases} E_{i,j} & \textit{si existe un lien local entre } \{i,j\} \\ 0 & \textit{dans le cas contraire} \end{cases}$$

Par exemple, si on considère que $E(x,y)$ est la latence entre deux noeuds x et y, et ρ est la différence maximale entre deux noeuds qui font partie du même groupe, l'algorithme le plus général pour cette approche considère que :

$$\forall x, \forall y \in S, x \neq y,\ a \in S \Rightarrow |E(a,x) - E(x,y)| \leq \rho$$

Le problème de cette approche est que pour chaque nouveau noeud, il faut comparer sa distance avec tous les autres noeuds qui sont à l'intérieur d'un groupe.

Une approche moins complexe est présentée par Lowekamp [LOW00] et utilisée dans la bibliothèque ECO (Algorithme 1). En effet, ECO utilise un algorithme glouton qui considère que :

$$\forall x, \forall y \in S, x \neq y,\ a \in S \Rightarrow |E(a,x) - min(E(x,y))| \leq \rho$$

Algorithme 1 Algorithme de Lowekamp [LOW00] pour le partitionnement du réseau ($\rho = 20\%$)

```
initialize subnets to empty
for all nodes
  node.min_edge = minimum cost edge incident on node
sort edges by nondecreasing cost
for all edges (a,b)
  if a and b are in the same subnet
    continue
  if edge.weight>1.20 * node(a).min_edge or edge.weight>1.20 * node(b).min_edge
    continue
  if node (a) in a subnet
    if (edge.weight>1.20 * node(a).subnet_min_edge)
      continue
  if node (b) in a subnet
    if (edge.weight>1.20 * node(b).subnet_min_edge)
      continue
  merge node(a).subnet and node(b).subnet
  set subnet_min_edge to min(edge,node(a).subnet_min_edge, node(b).subnet_min_edge)
```

Dans cette approche, chaque sous-réseau garde la valeur du plus petit arc à l'intérieur du groupe. Chaque nouvel arc est comparé avec cette valeur minimum, réduisant ainsi le nombre de comparaison à effectuer.

La contrepartie de l'algorithme de Lowekamp est qu'il permet certaines situations où des sous-réseaux hétérogènes sont formés. La Figure 3.1 présente un exemple de ce problème, où des noeuds hétérogènes sont incorrectement regroupés.

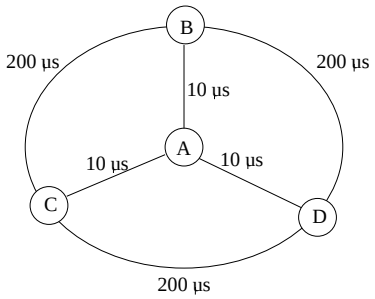


Figure 3.1 *Contre-exemple à l'algorithme de Lowekamp*

En effet, nous pouvons observer que le noeud A est le seul connecté aux autres avec une petite latence. Or, dans ce cas, un groupe {A, B, C, D} est possible, malgré le fait que l'interconnexion entre les noeuds B, C et D est beaucoup plus lente. Dans ce cas, l'hétérogénéité des connexions à l'intérieur du groupe peut influencer fortement les communications (par exemple, si B devient le coordinateur), sans que les modèles de communication soient capables de représenter les performances obtenues. Cependant, telles situations sont suffisamment rares pour permettre l'utilisation de l'algorithme de Lowekamp dans notre outil de découverte de topologie.

3.3 Troisième partie : acquisition efficace des paramètres d'interconnexion

Même si les grappes logiques identifiées par notre *framework* présentent la structure effective du réseau, nous ne sommes pas encore en mesure de modéliser les communications avec précision. La première raison est que la matrice de distances n'est pas totalement remplie. Comme expliqué dans la section 3.1, l'obtention des informations de connectivité se fait localement à chaque grappe, et les connexions entre les grappes ne sont pas encore établies.

Une autre raison est que les données obtenues par les outils de surveillance du réseau ne sont pas adaptées à nos modèles de communication. Par exemple, la latence est obtenue de façon différente par les outils NWS [WOL97] et LogP-MPI [KIE00]. Dans le cas de NWS, la latence est obtenue directement à partir du temps d'aller-retour d'un message. L'outil LogP-MPI, par contre, décompose le temps d'aller-retour en deux composants, la latence et le *gap*, comme illustré dans la Figure 3.2. Les différences entre les deux stratégies d'obtention de la latence sont suffisamment grandes pour interférer sur la précision des modèles de communication. De surcroît, les informations collectées par les outils de surveillance ne sont pas sujettes aux mêmes contraintes que les applications, comme par exemple l'utilisation d'un serveur Interoperable MPI - IMPI ([IMPI, SQU00]) pour interconnecter deux grappes séparées par un pare-feu.

Heureusement, nous n'avons plus besoin d'exécuter $n(n-1)$ mesures, une pour chaque interconnexion possible. En effet, l'homogénéité des grappes logiques permet qu'une seule mesure des paramètres d'interconnexion à l'intérieur de chaque grappe soit suffisante pour représenter toutes les communications à l'intérieur cette grappe. De même, il suffirait de mesurer une connexion entre chaque grappe pour représenter toutes les connexions distantes possibles.

Finalement, nous pouvons encore réduire le nombre de mesures par moitié, si nous considérons

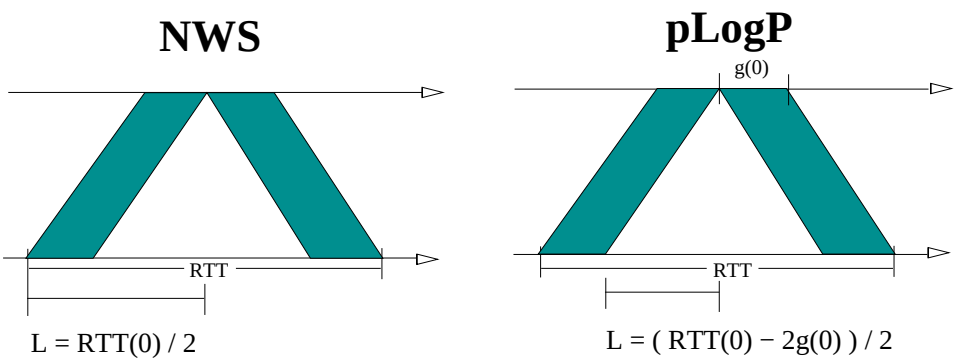


Figure 3.2 *Différence entre la latence de NWS et de pLogP*

que les liens de connexion sont symétriques $a \rightarrow b = b \rightarrow a$. Ainsi, se le nombre de grappes logiques identifiées précédemment est C, la quantité de mesures est réduite à :

$$\frac{C \times (C-1)}{2} + C$$

3.4 Évaluation pratique

Pour évaluer l'utilisation de notre *framework*, nous avons intégré les trois parties de découverte de topologie dans notre bibliothèque LaPIe. Les deux premières parties sont implantées en une application indépendante, alors que la troisième partie est implantée dans les procédures d'initialisation du MPI.

En effet, les étapes de collecte d'informations et de partitionnement des machines ont été regroupées en une application indépendante, exécutée par l'utilisateur avant le déploiement de son application, et qui fournit un fichier de description des grappes homogènes. Nous avons profité des fonctionnalités de la bibliothèque MagPIe, dont nous sommes inspirés, pour fournir à l'application une description de la répartition des machines qui sera utilisée ensuite pour la construction de la topologie des processus. Seule différence, notre bibliothèque a permis l'automatisation de la génération de ce fichier de description, ce qui est notamment avantageux du point de vue des heuristiques d'optimisation qui utilisent cette description.

La troisième partie a été intégrée à la fonction *MPI_Init*, appelée à l'initialisation de chaque application MPI. Nous avons intégré les fonctions d'obtention des paramètres d'interconnexion aux fonctions de description de la topologie. Cela permet notamment la prise en charge des coûts de communication réels de l'application.

Pour illustrer notre approche de découverte de topologie, notamment la première et la deuxième partie, nous présentons deux expériences, l'une sur la grappe IDPOT, et l'autre sur une grille de 78 machines réparties entre les sites d'Orsay, Toulouse, Sophia-Antipolis et Grenoble (IDPOT).

La grappe IDPOT est constituée de machines Intel Xeon biprocesseur dont une partie des machines utilise des cartes réseau Broadcom Giga Ethernet et l'autre partie utilise des cartes réseau Intel Giga Ethernet. L'interaction entre les différentes cartes réseau occasionne une forte variation de performance des communications, ce qui est très intéressante pour les expériences de découverte du réseau (conformément à ce que nous avons présenté en [BAR04b]).

À partir de la procédure de découverte de topologie présentée précédemment, nous avons pu repartir les machines IDPOT en trois groupes homogènes, comme indique la Figure 3.3.

```
cluster 0
process 0 1 2 3 6 7 8 9 11 12 17
cluster 1
process 4 5
cluster 2
process 10 13 14 15 16
# Min edge 0-0 = 35.524368
# Min edge 0-1 = 59.962273
# Min edge 0-2 = 59.962273
# Min edge 1-1 = 60.439110
# Min edge 1-2 = 122.54715
# Min edge 2-2 = 60.081482
```

Figure 3.3 *Fichier de description de grappes logique homogènes pour le réseau IDPOT, avec la latence entre les grappes (microsecondes)*

TAB. 3.1 *Gap pour un message de 1Mo sur les sous-réseaux IDPOT, en millisecondes*

	sous-réseau 0	sous-réseau 1	sous-réseau 2
sous-réseau 0	9,0906	19,7898	16,4515
sous-réseau 1	19,7898	25,0942	25,2701
sous-réseau 2	16,4515	25,2701	26,1203

Cette différence entre les machines IDPOT se reflète aussi sur le gap, comme indiqué dans le Tableau 3.1, malgré le fait que les machines appartiennent toutes au même sous-réseau IP, technique normalement utilisée pour regrouper les machines.

En effet, une telle variation des valeurs des paramètres d'interconnexion (le gap et la latence) ne permet pas une prédiction de performance très précise, car les modèles de performance considèrent que le réseau est uniforme, c'est-à-dire, une valeur unique de gap et latence pour tout le réseau. À l'aide de notre outil pour la découverte de topologie nous pouvons alors identifier les sous-réseaux homogènes et les traiter séparément, de manière, par exemple, à réorganiser les communications selon les heuristiques présentées dans le chapitre 7. En conséquence, l'identification des sous-réseaux homogènes rend possible une meilleure modélisation du réseau réel que le simple regroupement des noeuds par localisation ou sous-réseau IP.

La deuxième expérience réalisée considère une grille de 78 machines, dont 20 machines de la grappe GdX d'Orsay, 20 machines de la grappe de Toulouse, 19 machines de la grappe de Sophia-Antipolis et 17 machines de la grappe IDPOT.

À l'aide de notre application de découverte de topologie (appelée *subnets*), nous pouvons obtenir un fichier de description de la topologie similaire à celui présenté dans la Figure 3.4.

Il faut noter que les machines sont identifiées par le rang du processus MPI ; pour mieux identifier l'appartenance de chaque machine, nous présentons dans la Figure 3.5 la distribution des six grappes homogènes par rapport à la localisation de ces machines.

Dans cet environnement, l'hétérogénéité est majoritairement due au facteur géographique. Cependant, le facteur matériel est aussi important, comme illustré par les machines du réseau IDPOT. Dans ce cas, nous observons que certaines machines peuvent être regroupées en grappes différentes malgré leur proximité géographique. Ainsi, les latences d'interconnexion entre les différents sites sont présentées dans le Tableau 3.2. On observe que malgré le seuil de tolérance de 30% utilisée par l'algorithme de partitionnement, la différence entre les machines IDPOT est suffisamment élevée

```
cluster 0
process 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
cluster 1
process 20 21 22 23 26 28 29 30 32 33 38
cluster 2
process 24 25 27 31 34 35 37
cluster 3
process 36
cluster 4
process 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58
cluster 5
process 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77
```

Figure 3.4 *Fichier de description de grappes logique homogènes*

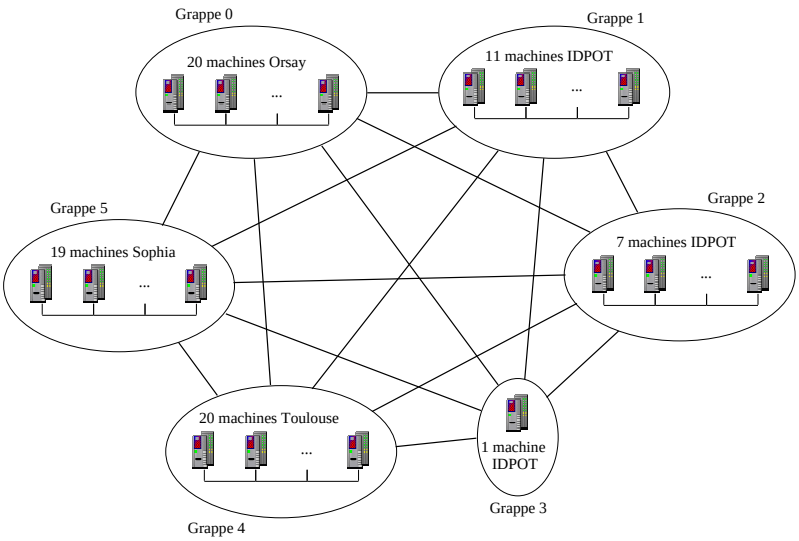


Figure 3.5 *Répartition des grappes homogènes dans une grille de 78 machines*

pour forcer le regroupement des machines en différentes grappes.

La dernière partie de notre *framework* de découverte de topologie est dédiée à l'obtention des paramètres d'interconnexion, nécessaires à la modélisation des communications. Cette partie est exécutée à l'initialisation de l'application, à partir des informations préliminaires fournies par le fichier de description de la topologie du réseau obtenu précédemment. Comme la procédure de mesure des paramètres peut demander un temps non négligeable, nous avons l'intérêt en minimiser le temps d'obtention des paramètres pour ne pas trop gêner l'utilisateur. À l'aide de notre *framework* de découverte de topologie, le nombre de mesures est naturellement réduit grâce au regroupement des machines en grappes logique homogènes. Dans le cas de notre exemple, le nombre de mesures nécessaires est seulement 21 ($((C \times (C-1))/2 + C$), ce qui représente moins de 1% du nombre total d'interconnexions ($(n \times (n-1)/2$). En plus, nous utilisons un algorithme d'ordonnancement pour que ces 21 mesures soient effectuées, dans la mesure du possible, de manière parallèle.

TAB. 3.2 *Latence entre les différents sites (en microsecondes)*

	Grappe 0	Grappe 1	Grappe 2	Grappe 3	Grappe 4	Grappe 5
	20 x Orsay	11 x IDPOT	7 x IDPOT	1 x IDPOT	20 x Toulouse	19 x Sophia
Grappe 0	48.39	6577.49	6586.49	6592.51	5211.94	8602.73
Grappe 1	6577.49	35.52	59.96	59.96	5387.48	2736.56
Grappe 2	6586.49	59.96	60.08	79.51	5393.98	2740.26
Grappe 3	6592.51	59.96	79.51	0*	5405.78	2745.98
Grappe 4	5211.94	5387.48	5393.98	5405.78	26.94	3630.51
Grappe 5	8602.73	2736.56	2740.26	2745.98	3630.51	35.04

* cette grappe contient une seule machine.

3.5 Extension des fonctionnalités : détection des noeuds SMP

Jusqu'au présent, nous avons considéré que la répartition des processus se fait dans l'ordre d'un processus par machine. Cela permet notamment une validation plus efficace des modèles de communication car il n'y a pas de concurrence pour les ressources d'un noeud. Néanmoins, il est quelque fois intéressant d'avoir plusieurs processus qui tournent sur une machine, notamment quand l'architecture lui permet (par exemple, il est conventionné que les machines qui font partie de la grille GRID5000 sont des biprocesseurs).

Étant donné la faible latence entre deux processus qui s'exécutent sur une même machine, la simple utilisation de l'algorithme de Lowekamp donnerait origine à une multitude de « sous-réseaux homogènes », composés par les processus qui tournent sur chaque machine. Dans ce cas-là, les efforts d'optimisation des communications collectives subissent une forte augmentation de complexité, car maintenant le coût de communication à l'intérieur d'un « sous-réseau » est très faible, et le nombre de « sous-réseaux » est proportionnel au nombre de machines connectées.

Afin d'optimiser les communications collectives dans le cas des machines multi-processeurs, nous avons besoin d'un découpage à deux niveaux : noeuds homogènes et processus sur un même noeud. Cependant, la distribution des processus dépend du déploiement de l'application, et pour cela, la détection des processus sur les noeuds SMP doit se faire pendant l'initialisation de l'application.

En effet, le déploiement des processus MPI se fait normalement de deux façons : par *round-robin* ou par CPU. Dans le premier cas, les processus sont distribués parmi les machines à l'aide d'une liste tournante, si le nombre de processus MPI est supérieur au nombre de machines disponibles. Dans le deuxième cas, les processus sont assignés selon le nombre de CPUs de chaque machine, avec un remplissage linéaire. Comme ces deux méthodes ne permettent pas la connaissance *a priori* de la répartition des processus, il faut que cette identification soit faite lors du démarrage des applications MPI.

Il reste néanmoins le problème de la détection de ces processus. Par définition, MPI identifie les processus par leur rang d'exécution et ne fournit pas une façon simple d'identifier quels processus tournent sur quelles machines. Par exemple, l'implantation SMP de LAM/MPI version 7 [LAM04] utilise des structures de données propriétaires de cette implantation de MPI pour identifier la localisation des processus.

Pour éviter la dépendance d'une implantation MPI spécifique, nous adoptons plutôt une approche plus générale. Au démarrage de l'application, chaque processus s'identifie avec un processus maître, qui se charge d'identifier les processus que tournent sur le même noeud. Pour cela, les processus utilisent l'appel *MPI_Get_processor_name()*, qui permet l'obtention du nom de la machine où ces

processus tournent. Cette alternative est similaire à l'appel du système *gethostname()*, mais son avantage est le fait que le nom de machine retourné est celui déclaré dans le fichier d'initialisation de MPI, évitant possibles confusions à cause de machines avec plusieurs noms ou noms incomplets.

Les informations communiquées au maître permettent donc l'établissement d'une carte de localisation des processus, qui peuvent alors être organisés en communicateurs et sous-communicateurs selon les grappes logiques identifiées précédemment. Bien sûr, il faut que les heuristiques d'optimisation soient capables d'utiliser ces informations de découpage à plusieurs niveaux. Dans ce cas-là, des travaux comme ceux Sun [SUN02] et Wu [WUM04] et l'expérience de l'implantation SMP de LAM/MPI version 7 [LAM04] peuvent être notamment utiles.

3.6 Discussion

Dans ce chapitre nous avons présenté une stratégie efficace pour identifier et découper les réseaux en grappes logiques homogènes. La présence des hétérogénéités réduit la précision des modèles de communication utilisés pour optimiser les communications collectives. Le *framework* proposé réussi à obtenir des paramètres de connectivité par un coût très faible, à partir du regroupement d'informations obtenues indépendamment dans chaque grappe. Notre approche, validée par des tests pratiques, démontre que la découverte de topologie peut se faire de façon rapide et précise. Associé à des modèles de prédiction de performance, le découpage des grappes s'avère essentiel à l'optimisation des primitives de communication collective, spécialement pour celles structurées en multiples couches. De plus, ce *framework* peut être étendu, de manière à détecter aussi la présence de machines multiprocesseur, information importante pour l'optimisation des communications.

Toutefois, la découverte de topologie a aussi des applications avec d'autres domaines que le calcul parallèle. En effet, nous avons travaillé au début de cette thèse avec l'application de la topologie sur les algorithmes repartis comme la Diffusion Totalement Ordonnée. Ainsi, l'Annexe A montre comme la connaissance de la topologie du réseau peut être utilisée pour augmenter la performance des algorithmes repartis, réputés peu performants.

II

Modélisation des Communications

4

Modélisation des Communication Point-à-Point

Dans le contexte de la modélisation de performance, nous appelons modèle la description formelle de l'exécution d'un programme sur une ou plusieurs machines, de manière à ce que cette description peut être utilisée pour comprendre, voire décrire, la performance de tel programme.

Il est clair que le principal objectif des algorithmes parallèles avec échange de message est la résolution de problèmes qui sont trop lents dans une machine séquentielle conventionnelle. Cependant, la simple répartition des tâches n'est pas une garantie de performance et la recherche d'algorithmes performants passe nécessairement par la compréhension des caractéristiques de ces algorithmes et de l'environnement d'exécution.

Une des meilleures manières de comprendre le fonctionnement des algorithmes parallèles et d'évaluer leur efficacité est de modéliser la performance de ces algorithmes. Si d'un côté il existent certains facteurs non déterministes qui influencent la performance des applications parallèles, comme par exemple la congestion des ressources, pour la plupart du temps son impact sur le temps d'exécution des applications est suffisamment limité [GRO03]. Ainsi, la plus grande difficulté pour la modélisation des performances est la correcte représentation des facteurs liés à la communication entre les processus répartis. Pour cette raison, la plupart des modèles de performance préfèrent décrire les systèmes de la manière la plus simple possible ; la perte de précision est compensée par la simplicité et par la portabilité des solutions à travers diverses situations et environnements.

Si les principes de la modélisation de performance peuvent être trouvés dans des travaux pionniers des années 60 et 70 [PEJ81], la modélisation de performance a connu une importante attention à partir des années 80, où des efforts importants ont été faits pour identifier des modèles de performance adaptés aux technologies et paradigmes qui ont surgi depuis la décennie précédente. Pour cette raison, l'objectif de ce chapitre est de présenter un résumé des modèles de performance les plus utilisés actuellement pour la modélisation des communications dans un environnement à mémoire distribuée. Cela sert non seulement à situer l'évolution des modèles dans un contexte historique, mais surtout à identifier les modèles qui ont les caractéristiques les plus adaptées à nos besoins.

4.1 PRAM

Le modèle PRAM (*Parallel Random Access Machine*) a été proposé par Fortune et Wyllie [FOR78] comme un modèle de performance pour le calcul parallèle sur des machines à mémoire partagée. Le modèle PRAM est composé par des processeurs associés à une mémoire locale. La communication entre les processeurs se fait à travers une mémoire partagée dont chaque événement de communication (envoi ou réception) est effectué en une seule unité de temps.

D'autre part, chaque processeur exécute une instruction en une unité de temps et tous les processeurs exécutent le même nombre d'instructions (même si les instructions peuvent varier) sur les différentes données de manière synchrone.

L'ample acceptation du modèle PRAM est surtout due à sa simplicité et à sa généralité. À cause de cette acceptation, un grand nombre d'algorithmes parallèles ont été développés sur ce modèle ; plusieurs techniques et paradigmes de programmation ont été fondés sur le modèle PRAM.

Cependant, le modèle PRAM n'est pas adapté à la modélisation de performance dans un environnement de type mémoire partagée virtuelle ou mémoire distribuée. En effet, ce modèle n'est pas capable de distinguer le temps d'accès aux différents machines, une fois qu'il considère un temps d'accès constant. Alors que plusieurs alternatives pour adapter ce modèle aux systèmes hétérogènes ont été proposées (voir le travail de Boeres [BOE96]), le modèle PRAM reste néanmoins fortement attaché aux systèmes à mémoire partagée.

4.2 Hockney

Le modèle de Hockney [HOK94] est un des plus utilisés pour décrire la communication point-à-point dans les machines parallèles à mémoire distribuée. Selon ce modèle, une communication entre deux noeuds est décrite par :

$$t = t_0 + \frac{m}{r_\infty}$$

où t_0 représente le temps nécessaire à l'envoi d'un message de taille zéro, m est la taille du message (en octets) et r_∞ est le débit asymptotique en Moctets/s, i.e., le débit maximal obtenu quand la taille du message s'approche de l'infini. Dans ce raisonnement, m/r_∞ représente le délai de transmission d'un message de m octets à travers un réseau avec un débit asymptotique de r_∞ Moctets/s.

Ce modèle est souvent transformé dans l'équation affine

$$t = \alpha + \beta m$$

où α correspond à la latence de transmission et β est le temps de transfert d'un octet sur ce réseau [PJE05]. L'obtention des paramètres α et β peut se faire à travers l'utilisation de certains outils comme NWS [WOL97]. Alors que la procédure pour obtenir ces paramètres a déjà été expliqué en chapitre 2, nous faisons aussi en section 3.3 une analyse sur la précision de cette méthode de mesure par rapport aux modèles de performance développés dans ce travail.

En effet, un inconvénient de ce modèle est qu'il assume que le temps de transfert est proportionnel à la taille du message. Dans des situations réelles, certains facteurs comme la taille de la mémoire tampon et la segmentation des messages en paquets font varier le temps de transfert β selon la taille du message. Cette constatation, qui a aussi donnée origine aux modèles LogGP et pLogP (décrits dans les pages suivantes), a été initialement décrite par Stoica *et al.* [STO96], qui a proposé un modèle adaptative appelé *Hyperbolique.* Dans ce modèle, le coût d'une transmission est défini par une fonction hyperbolique de type

$$t = \frac{a^2}{a + bm} + bm$$

où a correspond au surcoût d'initialisation d'une transmission (latence incluse) et b au débit asymptotique de la transmission. De cette manière, le modèle permet au temps de transfert de varier selon la taille du message, avec une dominance du surcoût d'initialisation quand les messages sont petits ($m \to 0$) et du débit de transfert quand les messages sont grands ($m \to \infty$). Cependant, la définition des paramètres a et b adaptés à chaque réseau augmente la complexité du modèle et pour cette raison ce modèle n'a pas été aussi répandu que les modèles LogGP et pLogP.

4.3 Postal

À partir de l'observation qu'un des principaux paramètres pour la modélisation des algorithmes parallèles est le coût de communication entre les processus, Bar-Noy et Kipnis ont proposé le modèle Postal [BAO94]. Le modèle Postal est fondé sur un paramètre $\lambda = t_u / t_{snd}$, où t_{snd} est le temps nécessaire à un processus pour envoyer un message (la latence d'envoi), alors que t_u représente le temps total nécessaire à la réception du message par le processus destinataire. Une des innovations du modèle Postal par rapport à ses prédécesseurs est que ce modèle permet des situations où un processus peut émettre plusieurs messages avant que le premier récepteur a reçu son message : cela est dû à une analogie avec l'envoi de lettres par la poste.

Prenons par exemple le cas d'une diffusion ou *broadcast*. Par simplicité, il est considéré que λ est une valeur entière et que le temps est mesuré en cycles, dont la durée est équivalente à la latence d'envoi d'un message. Dans le cas du *Broadcast*, la valeur de λ influence la forme de l'arbre de diffusion qui permet la meilleure performance. C'est ainsi que la Figure 4.1 compare deux arbres de diffusion différents, chacun contenant huit noeuds et $\lambda = 2$. On observe que l'arbre binomial, (Figure 4.1a), malgré son optimalité quand $\lambda = 1$, nécessite six étapes de communication pour compléter l'opération, alors que l'arbre représenté dans la Figure 4.1b nécessite seulement cinq étapes. En effet, la construction de l'arbre de diffusion pour P processus avec le modèle Postal (un arbre-λ) est fondée sur la fonction de Fibonacci généralisée :

$$N_\lambda(t) = \begin{cases} N_\lambda(t-1) + N_\lambda(t-\lambda) & si\ t \geq \lambda, \\ 1 & dans\ le\ cas\ contraire \end{cases}$$

Dans ce cas, $N_\lambda(t)$ représente le nombre maximum de noeuds qui peuvent être contactés à l'instant t sur une architecture réseau *1-port*. Si $t < \lambda$, seul le noeud source peut détenir le message à l'instant t. Par contre, si $t \geq \lambda$, le nombre de noeuds contactés est la somme de deux éléments. Le premier élément, $N_\lambda(t-1)$, indique le nombre de noeuds qui ont été contactés jusqu'à l'instant précédent. Le deuxième facteur, $N_\lambda(t-\lambda)$, indique le nombre maximum de noeuds qui peuvent être contactés à l'instant t.

4.4 BSP

Le modèle BSP (*Bulk-Synchronous Parallel*) a été proposé par Valiant [VAL90] comme un modèle de performance capable de capturer les caractéristiques les plus remarquées des architectures parallèles et en même temps d'abstraire les caractéristiques du réseau, qui ne sont pas standardisées. C'est ainsi qu'une machine abstraite BSP consiste en un groupe de processeurs, un réseau de communication et un mécanisme efficace de synchronisation (barrière) entre tous les processus.

L'une des principales caractéristiques du modèle BSP est le concept de super-pas (*superstep*), qui est défini comme une séquence d'opérations locales suivie par une synchronisation. L'exécution d'une application BSP est une succession de super-pas, dont chacune consiste en tâches de calcul et de communication entre les noeuds. Pendant un super-pas, toutes les données nécessaires à l'exécution des tâches doivent être localement disponibles : toute donnée nécessaire doit être préalablement obtenue des mémoires non-locales. Une fois que tous les processeurs ont exécuté leurs

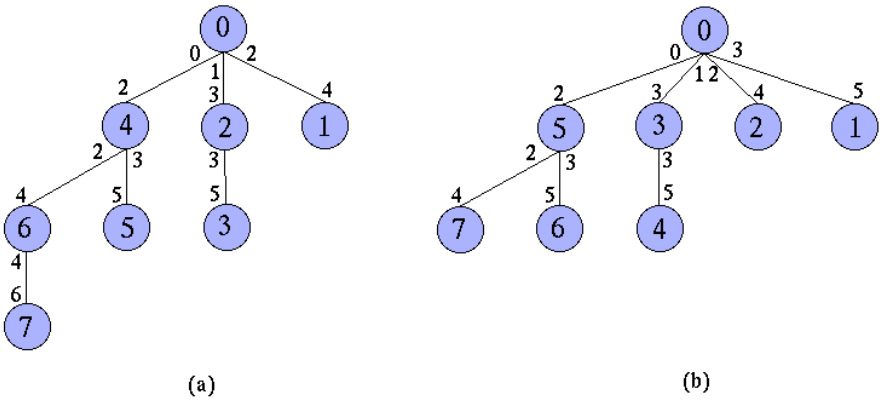


Figure 4.1 *Broadcast avec $\lambda = 2$:arbre binomial et arbre-λ*

super-pas respectifs, une barrière de synchronisation est exécutée de manière à permettre qu'un nouveau super-pas soit initié. La Figure 4.2 illustre l'exécution d'un super-pas entre deux barrières de synchronisation. Pendant le super-pas, tous les calculs et communications nécessaires au super-pas sont exécutés avant la deuxième barrière de synchronisation.

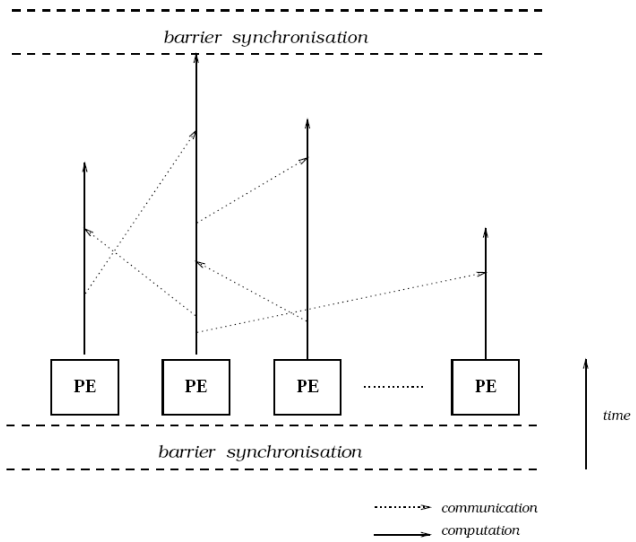


Figure 4.2 *Un super-pas selon le modèle BSP*

Dans le modèle BSP, ni l'ordre d'exécution des tâches ni l'ordre de délivrance des messages est prédéfinie. Pour cette raison, le parallélisme des tâches peut être mieux exploité, une fois qu'une transmission de messages peut être recouverte par l'exécution de tâches locales dont les données sont déjà disponibles.

Pour représenter le coût des super-pas, le modèle BSP est caractérisé par les paramètres suivants :

P - le nombre de processeurs ;

s - la vitesse des processeurs, définie par le nombre d'opérations de point flottant par seconde ;

l - l'intervalle minimal de temps entre deux synchronisations ;

g - la relation entre le nombre total d'opérations exécutées par le processeur et le nombre de « mots » délivrés par le réseau, par unité de temps.

Le paramètre g représente le coût global de communication, incorporant la latence et la congestion du réseau. Si on suppose que chaque processeur utilise le même patron de communication et qu'il peut envoyer et recevoir jusqu'à h unités de données, la limite supérieure de communication globale dans le modèle BSP est donc de gh unités de temps.

La taille minimale d'un super-pas est le temps minimal entre deux synchronisations globales successives différentes, représentée par l. La taille maximale d'un super-pas est une propriété de l'application qui dépend de la quantité de travail exécutée entre deux points de synchronisation successifs, mais aussi du degré de parallélisme de l'application.

Le modèle BSP permet la représentation des principales caractéristiques de communication du réseau à travers les paramètres l et g, étant plus efficace pour les machines dont les paramètres l et g sont petits, de manière à ce que le temps nécessaire à la complétion des calculs soit minimal. En effet, si on défini $l = g = 1$, le modèle BSP est équivalent au modèle de performance PRAM. Toutefois, le paramètre g généralement dépend de plusieurs caractéristiques du système, comme par exemple le nombre de processeurs, leur vitesse et aussi le débit du réseau.

Si le modèle BSP a pour objectif la proposition d'une structure unifiée de calcul et une représentation portable des performances sur les différentes architectures parallèles, ce modèle n'a pas la capacité suffisante pour faire l'abstraction nécessaire pour la représentation de certaines machines comme les grappes d'ordinateurs. En effet, le modèle BSP n'inclut pas les informations sur le surcoût d'initialisation des communications, important sur ce type de machines. Le modèle BSP assume que les messages échangés entre les processus sont suffisamment grands pour que le surcoût d'initialisation soit ignoré. Cependant, même si l'on considère que des messages longs, le fait d'ignorer les détails de la topologie d'interconnexion peut influencer fortement l'exécution d'une application dans un tel environnement.

Par ailleurs, les machines parallèles ne se comportent pas toujours comme le modèle BSP spécifié. Par exemple, des communications non-bloquantes comme celles à l'intérieur des super-pas peuvent ne pas être possibles, selon l'implantation utilisée. De la même manière, le mécanisme de synchronisation peut être très cher à implanter. En effet, quand la synchronisation dépend du système de communication, le réseau doit offrir des stratégies pour synchroniser tous les processus, ce qui peut avoir un coût très élevé.

Cette difficulté du modèle BSP est dû à l'utilisation d'un seul paramètre g pour représenter plusieurs aspects de l'architecture d'intercommunication, comme par exemple, le débit, la latence et la congestion. Étant liée aux caractéristiques du réseau, la latence peut avoir une valeur bien plus élevée que le modèle considère normalement ; de plus, la latence de communication ne peut pas être toujours recouverte par des calculs.

4.5 LogP

Malgré le développement de processeurs et de réseaux plus efficaces, la communication efficace entre ces deux éléments a toujours été un obstacle à l'obtention de meilleures performances. En effet, l'augmentation du nombre de processeurs ne permet pas nécessairement que le temps d'une application soit réduit proportionnellement : en dehors des limitations dues au grain des tâches, le coût de communication entre les différents processeurs est l'un des plus importants obstacles.

Dans une tentative de spécifier un modèle de calcul plus réaliste, Culler *et al.* [CUL96a] ont proposé le modèle de coût LogP. Alors que le modèle BSP a servi d'inspiration pour le modèle LogP, il ne permettait pas la spécification du surcoût d'envoi, qui est normalement très significatif pour la performance d'un système. L'importance de ce paramètre est notamment observée sur des expériences en machines réelles et, surtout, dans le cas des grappes d'ordinateurs.

Une autre caractéristique qui différencie les modèles LogP et BSP est le comportement asynchrone de LogP. En effet, le modèle LogP ne requiert ni barrière de synchronisation, ni super-pas. Quand un processus finit l'exécution d'un groupe de tâches et de communications, il peut automatiquement avancer vers le prochain groupe de tâches, même s'ils existent encore des processeurs travaillant sur une tâche précédente. En contraste, le modèle BSP forçait le processeur à rester à l'attente de la barrière de synchronisation. Ainsi, ce comportement asynchrone du modèle LogP peut être considéré comme un avantage, notamment si l'application permet l'avancement asynchrone des différentes lignes d'exécution.

Bien sûr, LogP reste un modèle simplifié des architectures parallèles, dont certains aspects de la topologie du réseau, comme par exemple la congestion, ne sont pas directement considérés (malgré la possibilité de travailler avec des valeurs de g qui correspondent à des communications sur des liens saturés). Les paramètres LogP qui caractérisent la communication entre les processus sont les suivants :

L - représente la latence de communication entre deux processus distincts,

o - le surcoût d'initialisation associée à l'envoi/réception d'un message,

g - le temps minimal nécessaire entre deux événements consécutifs d'envoi ou de réception,

P - le nombre de processus.

Sous cette représentation, le modèle Postal de Bar-Noy et Kipnis devient un cas spécial du modèle LogP (avec $g = 1$ et $o = 0$).

Dans le cas du modèle LogP, le nombre maximum de messages en transit entre deux processus est de $\lceil L/g \rceil$. La latence maximale d'un réseau est la distance moyenne asymptotique entre les processus ; toutefois, dans le cas des réseaux fortement hétérogènes, la définition de ces limites est bien plus complexe.

Pour mieux illustrer l'utilisation des paramètres définis par LogP, nous présentons dans la Figure 4.3 l'exemple proposé par Culler où LogP est utilisé pour décrire le broadcast optimal sur un réseau avec $P = 8$, $L = 6$, $g = 4$ et $o = 2$. Dans cet exemple, nous observons qu'une communication isolée nécessite $o + L + o$ unités de temps pour être reçue ; cependant, la transmission consécutive de deux messages implique un surcoût équivalent à $(g - o)$, car on n'est pas autorisé à transmettre deux messages consécutives en moins de g unités de temps. En effet, le temps o correspond au temps nécessaire au traitement du message pour la transmission (encapsulation, mise en mémoire tampon, etc.), alors que le gap g correspond à l'intervalle où le réseau reste occupé par la transmission du message.

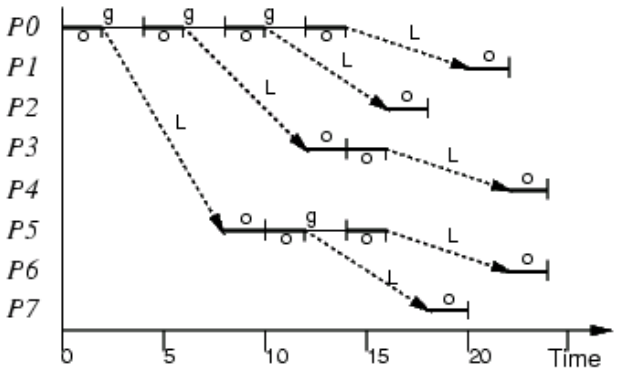


Figure 4.3 *Diffusion en LogP avec P=8, L=6, g=4 et 0=2 [CUL96a]*

4.6 LogGP

Comme le comportement asynchrone de plusieurs machines parallèles et le surcoût de communication sont représentés par le modèle LogP, ce modèle est bien plus complet que les modèles précédents. Cependant, la faiblesse de ce modèle est que ses paramètres sont établis de façon absolue : les valeurs de g et o sont les mêmes pour n'importe quelle taille de message envoyé. Des expériences pratiques démontrent que l'utilisation de LogP n'est précise que pour des messages de petite taille, dont g et o ne varient pas trop.

Pour permettre l'utilisation du modèle LogP avec des messages plus grands, Alexandrov *et al.* [ALE95] ont présenté LogGP, une extension du modèle LogP qui établit le paramètre G, qui représente une valeur de *gap* par octet transmis. À travers ce nouveau paramètre, LogGP permet la prédiction du temps de transmission d'un message de taille m entre deux noeuds avec l'expression :

$$L + 2 \times o + (m-1) \times G$$

4.7 pLogP

Comme LogGP, le modèle pLogP (*parameterised LogP*) est une extension du modèle LogP. Présenté par Kielmann *et al.* [KIE01], le modèle pLogP propose l'unification des modèles LogP et LogGP pour mieux représenter, à la fois, des petits et des grands messages.

Comme ses prédécesseurs, le modèle pLogP est défini à travers des paramètres qui représentent la latence, les surcoûts d'initialisation, le *gap* et le nombre de processus. La première différence est l'utilisation de valeurs distinctes, o_r et o_s, pour représenter le surcoût d'envoi et le surcoût de réception d'un message. Toutefois, la principale caractéristique du modèle pLogP est que les paramètres qui représentent le *gap* et les surcoûts sont paramétrés selon la taille du message envoyé. Cela est spécialement important pour modéliser les communications avec des tailles de messages variables, une fois que, comme déjà observé par Alexandrov [ALE95], la performance des communications n'est pas linéaire par rapport à la taille des messages.

D'autres aspects sont aussi différents, par rapport aux modèles précédents. En effet, les notions de la latence et du *gap* sont légèrement différentes de celles utilisées par les modèles LogP/LogGP. Dans le cas du modèle pLogP, la latence inclut tous les facteurs qui peuvent retarder la communication entre deux processus, comme par exemple la copie de données en mémoire tampon vers les interfaces réseaux, qui s'ajoutent au temps de transfert des messages déjà considéré par LogP. Ainsi, dans le cas d'un réseau local, le paramètre *gap* est défini comme l'intervalle minimale entre deux transmissions ou réceptions consécutives, ce qui implique que $g(m) \geq os(m)$ et $g(m) \geq or(m)$ tient toujours. La relation entre ces différents paramètres est illustrée dans la Figure 4.4.

En conséquence de cette nouvelle interprétation du *gap*, les paramètres o_s et o_r ont une importance moins évidente, une fois que leur coût dans un réseau local est souvent recouvert par celui du gap. En effet, lors d'une communication bloquante (par exemple, avec MPI_Send et MPI_Recv) seulement le *gap* et la latence sont visibles lors du calcul du coût d'une connexion, alors que les paramètres o_s et o_r deviennent importantes dans le cadre des communications non-bloquantes (MPI_Isend et MPI_Irecv), où ces paramètres représentent le moindre temps nécessaire à la mise en mémoire tampon d'un message pour la transmission postérieure ou de sa lecture à partir de la mémoire tampon.

Ainsi, pour représenter le temps nécessaire à la transmission d'un message de taille m entre deux noeuds avec des primitives de communication bloquante, le modèle pLogP utilise l'expression $L + g(m)$, au lieu de $L + g + 2o$ comme dans le modèle LogP.

Il est aussi nécessaire d'ajouter que la modélisation des performances de communication pour un réseau local diffère un peu de la modélisation pour une grille de calcul. En effet, selon le comportement du protocole réseau et des commutateurs, la communication à longue distance peut s'échelon-

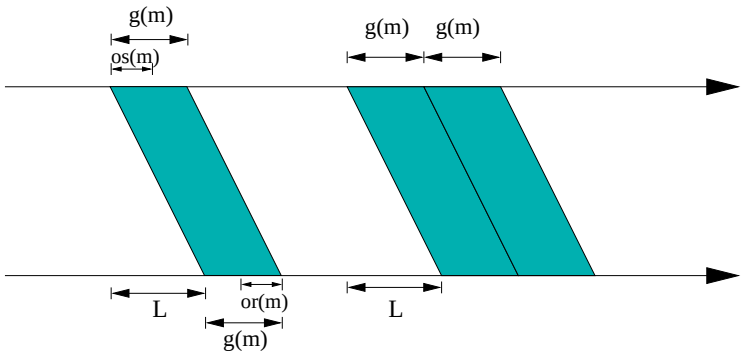


Figure 4.4 *Représentation d'une communication avec pLogP*

ner en plusieurs étapes qui sont concurrentes avec la communication sur le réseau local. Ainsi, par exemple, Kielmann [KIE01] a considéré une communication à travers une *passerelle* (*gateway* en anglais) qui relaye les messages vers des machines distantes. Dans ce cas spécifique (par exemple, avec un commutateur *store and forward*), l'émetteur n'est pas tenu d'attendre la transmission des messages jusqu'à la machine distante, pouvant donc initialiser la communication vers d'autres processus dans un temps $max(g_l(m), os_w(m))$ - où $g_l(m)$ indique le gap du réseau local et $os_w(m)$ indique le surcoût d'envoi vers le réseau distant. Cependant, dans le cas où les messages ne sont pas relayés (comme dans le cas des commutateurs *cut-through*), c'est le coût total de transmission vers le lien distant qui doit être considéré, de telle manière à ce que l'émetteur reste bloqué pendant $max(g_l(m), g_w(m))$.

4.8 Choix du modèle de coût

Alors que les modèles de performance présentées dans ce chapitre ont différentes niveaux de complexité et de précision, il est important de définir celui qui est le plus adapté aux besoins de nos expériences. Dans cet aspect, nous pouvons classer les modèles en trois catégories, celle des modèles avec coût de communication fixe (BSP et LogP), celles avec un coût de communication linéaire (Hockney et LogGP) et celles avec coût de communication variable (Hyperbolique, pLogP).

La première catégorie considère que le coût de transmission d'un message est unique pour n'importe quel taille de message envoyé. En réalité, les raisons pour laquelle les deux modèles, BSP et LogP, utilisent un coût fixe sont différentes : BSP considère que le coût g est fixe car il représente le coût de communication d'un super-pas, alors que LogP utilise une valeur fixe qui serait intrinsèque à chaque réseau.

Dans la deuxième catégorie nous trouvons les modèles Hockney et LogGP, où le coût d'envoi d'un message est proportionnel à la taille du message envoyé : les paramètres β ou G représentent alors le coût par octet envoyé.

Finalement, le dernier groupe considère que le coût d'envoi d'un message varie selon le nombre d'octets envoyés, mais qui ce coût n'est pas toujours proportionnel. En effet, le modèle Hyperbolique considère qu'il existe une transition entre l'envoi de petits et de grands messages, ce qui peut être considéré comme équivalent à l'utilisation « mixte » des modèles LogP et LogGP. Dans le cas du modèle pLogP toutefois le coût associé à une taille de message peut être encore plus variable - une pratique courante est d'utiliser comme référence un tableau avec des échantillons de mesure et d'inférer le coût d'un message donné à travers l'interpolation de deux mesures proches.

La question alors est d'identifier si les communications d'une application MPI seront soumises à des variations de performance. Un premier indice est la politique de communication employé par

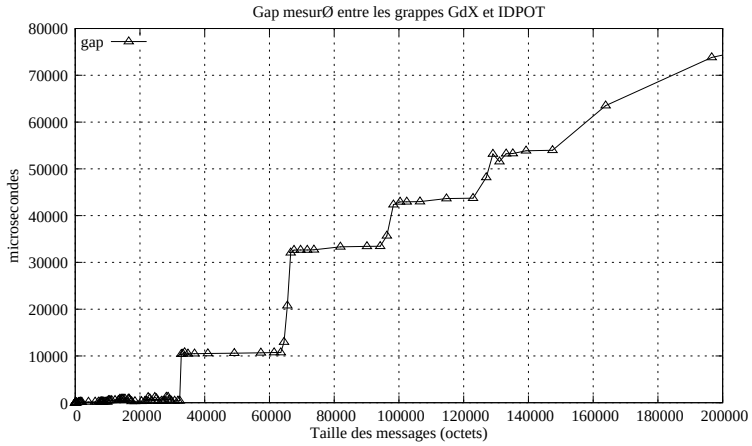


Figure 4.5 *Valeurs de gap mesurés entre deux grappes distantes (GdX et IDPOT)*

MPI. En général, des messages petits sont envoyés directement aux destinataires, alors que les messages plus volumineux subissent un protocole de *rendez-vous* pour s'assurer que le destinataire est prêt à recevoir le message. Ces différentes politiques de transmission ont obligatoirement un impact sur la linéarité du coût de communication. Dans le cas de la bibliothèque LAM-MPI [LAM04], utilisée dans nos expériences, le changement de politique se fait quand la taille des messages dépasse les 64 Ko. Cette variation de performance est illustré dans la Figure 4.5, où le coût du gap dépend à la fois de la saturation de la fenêtre TCP et du changement de politique d'acquittement de LAM.

Le différent comportement des communications selon la taille des messages, allié à la variété de patrons de communication (Broadcast, Scatter, Reduce, ...), favorise l'utilisation d'un modèle de coût plus général et adapté au modèle de communication 1-port. Ceci est également mis en évidence lorsque la technique de segmentation est utilisé, car la taille des segments généralement correspond à des messages de petite taille, très sujets aux variations observées dans la Figure 4.5.

Vu le caractère général du modèle pLogP, celui-ci semble le plus approprié pour modéliser les performances des communications collectives. Ce même constat a été fait par Pjesivac-Grbovic et Dongarra [PJE05], qui ont comparé la précision des modèles Hockney, LogP, LogGP et pLogP lors de la modélisation de l'opération MPI_Bcast. Leur analyse indique que le modèle pLogP est le plus précis pour la modélisation des communications collectives, notamment à cause de sa flexibilité pour le maniement de messages de différentes tailles.

4.8.1 Obtention des paramètres pLogP

Étant le modèle choisi pour représenter les différents algorithmes de communication collective dans notre travail, il est aussi important de bien obtenir les paramètres nécessaires à ce modèle.

Pour cela, nous avons utilisé la méthode proposée et implantée par Kielmann [KIE00]. L'avantage de cette méthode par rapport à des techniques précédentes comme celles de Culler [CUL96b] et Ianello [IAN98] réside dans l'obtention du paramètre *gap*. En effet, les méthodes décrits par Culler et Ianello utilisent des grands messages pour saturer le réseau, afin que le débit du lien (représenté par le *gap*) soit mesuré. Cependant, cette technique est très intrusive et nécessite un temps assez élevé pour mesurer le gap dans les réseaux à très haut débit.

Dans l'approche proposé par Kielmann, par contre, le gap est obtenu à partir de l'envoi de petits messages de taille zéro. Ainsi, pour mesurer *g(0)*, la méthode proposé mesure le temps d'aller-retour d'un message de taille zéro - $RTT_1(0)$ - entre deux processus, *measure* et *mirror*. Ensuite, le nombre

de messages envoyés simultanés est augmenté, de manière à ce que le temps $RTT_n(0)$ indique le temps nécessaire à l'envoi de n messages vers le *mirror* suivies d'une unique réponse d'acquittement vers le *measure*. Le nombre de messages n est doublé jusqu'à ce que le *gap* par message ne varie que de d'une marge d'erreur ϵ. À ce point, $RTT_n(0)$ est considéré comme équivalent à $n \times g(0)$ et la valeur de $g(0)$ peut être facilement obtenue.

En effet, un premier échange envoie un message de m octets, qui est répondu par un message de taille zéro (c'est équivalent à faire $RTT_1(m)$). À partir de $RTT_1(m)$, les valeurs de $g(m)$ et L peuvent être déduites directement à partir des équations présentées dans le Tableau 4.1.

TAB. 4.1 *Équations pour l'obtention de $g(m)$ avec pLogP [KIE00]*

$$\begin{array}{c} RTT_1(0) = 2 \times (L + g(0)) \\ L = (RTT_1(0) - 2 \times g(0))/2 \\ RTT_1(m) = L + g(m) + L + g(0) \\ g(m) = RTT_1(m) - RTT_1(0) + g(0) \end{array}$$

Pour obtenir les autres paramètres comme $o_s(m)$ et $o_r(m)$, nous utilisons la procédure représentée dans la Figure 4.6. Dans un premier moment, le paramètre $o_s(m)$ est obtenu à partir de la mesure du temps de l'envoi simple (sans attendre une réponse). Dans le deuxième échange de messages, le processus *measure* envoie un message de taille zéro et attend pendant un temps $\Delta > RTT_1(m)$ avant de recevoir un message de m octets. Cet échange permet l'obtention de $o_r(m)$, car après le temps $\Delta > RTT_1(m)$ le processus *measure* est certain que le message envoyé par *mirror* est déjà disponible dans la mémoire tampon de son interface réseau. Il faut observer cependant que cette mesure de $o_r(m)$ n'est correcte que si la communication entre les deux noeuds *measure* et *mirror* est symétrique. Si le débit entre *mirror* et *measure* est moins important que celui entre *measure* et *mirror*, cela peut donner des valeurs de $o_r(m)$ plus grands que $g(m)$. Ainsi, une manière d'éviter cette situation serait de mesurer $o_r(m)$ dans une communication *measure-measure*, si $o_r(m)$ est un paramètre essentiel aux modèles étudiés.

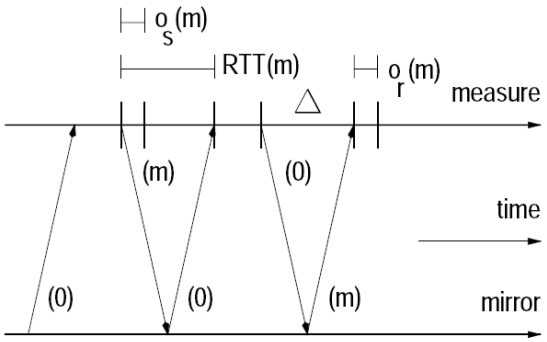


Figure 4.6 *Méthode d'obtention des paramètres pLogP [KIE00]*

Répété avec différentes valeurs de m, cette procédure permet l'obtention de $g(m)$, $o_s(m)$ et $o_r(m)$ à partir d'un tableau contenant une liste d'échantillons, comme présenté dans la Figure 4.7. Ces échantillons sont obtenus de manière à ce que la détermination des valeurs pour une taille de message m donnée soit faite à partir de l'interpolation des points de mesure les plus proches.

```
# LogP network performance data: logp_test.Send.Recv
# Latency = 32.66
# time bytes os os_min os_cnfint or or_min or_cnfint g
1113829432 0 0.0000053 0.0000043 0.0000004 0.0000024 0.0000014 0.0000009 0.0000014
1113829432 1 0.0000056 0.0000043 0.0000005 0.0000036 0.0000024 0.0000004 0.0000035
1113829432 2 0.0000058 0.0000043 0.0000004 0.0000038 0.0000033 0.0000010 0.0000036
. . .
1113829432 256 0.0000059 0.0000043 0.0000005 0.0000042 0.0000033 0.0000007 0.0000090
1113829432 512 0.0000061 0.0000045 0.0000011 0.0000041 0.0000033 0.0000009 0.0000158
1113829432 1024 0.0000063 0.0000052 0.0000010 0.0000044 0.0000033 0.0000004 0.0000292
. . .
1113829432 262144 0.0017727 0.0017314 0.0001001 0.0023521 0.0023274 0.0001276 0.0023500
1113829433 524288 0.0039728 0.0038664 0.0002125 0.0046024 0.0045595 0.0002578 0.0045943
1113829433 1048576 0.0085095 0.0084665 0.0002588 0.0090407 0.0090144 0.0001601 0.0091020
. . .
```

Figure 4.7 *Exemple d'un tableau avec les paramètres pLogP pour des tailles de message différentes (1 octet jusqu'à 1Mo)*

4.8.2 Application pratique : mesure du débit d'un lien de longue distance

Un problème courant dans les environnements de type grille de calcul est la nécessité de redistribution de données entre deux grappes distantes. Pendant l'exécution d'une application, des noeuds appartenant à des grappes différentes échangent des données, sans avoir connaissance des communications concurrentes. À ce moment, l'efficacité des transferts dépend de la capacité de contrôler le partage du débit, afin d'éviter les problèmes liés à la congestion du réseau et de maximiser l'utilisation du débit tout au long du réseau d'interconnexion, comme indiquent les travaux de Wagner *et al.* [WAG04] et Burger *et al.* [BUR05].

Pour que le partage du débit soit fait de manière vraiment efficace, les applications doivent avoir la connaissance des capacités réelles du lien d'interconnexion entre les deux grappes. Malheureusement, les outils de mesure de performance des communications traditionnelles (NWS [WOL97], par exemple) ont une vision plus proche des couches de transport et ne tiennent compte des limitations imposées par la couche d'application (MPI, par exemple). Ainsi, le débit maximal atteint par une application est parfois beaucoup plus inférieur à celui mesuré par NWS.

De ce fait, l'évaluation des capacités effectives des liens de très haut débit devient un problème relativement complexe à résoudre, même dans le cas des outils comme le NWS. Par exemple, l'évaluation du débit maximal d'un lien nécessite la saturation de ce lien et une seule paire de noeuds n'est pas capable de saturer un lien à très haut débit (le VTHD[1] a la capacité estimée à 2,5 Gbit/s, alors que la capacité maximale d'un seul noeud est généralement à moins de 1 Gbit/s).

Au lieu d'augmenter le débit atteint par les noeuds pour évaluer la capacité des liens d'interconnexion, la solution la plus simple passe par la multiplication des sources de données, de manière à saturer le lien à haut débit à travers l'agrégation de plusieurs connexions simultanées.

C'est ainsi que nous avons établi une procédure de mesure du débit agrégé pour évaluer les liens de longue distance, laquelle a été utilisée dans le travail de DEA de Carlos Barrios Hernandez au Laboratoire ID-IMAG. À partir de l'algorithme de mesure des paramètres pLogP de Kielmann [KIE00], nous avons développé une application qui met en communication plusieurs noeuds comme illustré par la Figure 4.8, de manière à identifier la capacité maximale du réseau.

[1] http ://www.vthd.org

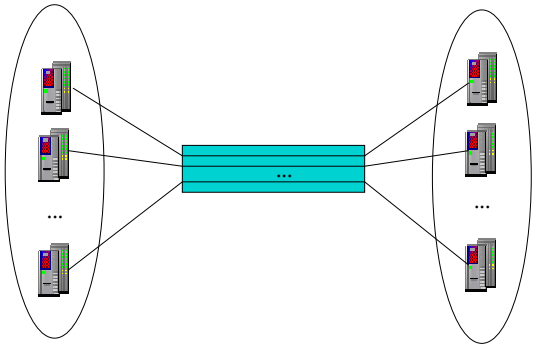


Figure 4.8 *Approche utilisée pour la saturation du lien de haut débit*

Pour cela, chaque paire de processus exécute l'algorithme de LogP-MPI [KIE00]. Le nombre de paires est augmenté progressivement, de manière à ce que le lien de communication soit progressivement saturé. L'équité du partage du débit entre les différentes connexions est assurée par la propriété *fairness* du protocole de transport TCP.

Avec l'augmentation du nombre de paires communicantes, le débit est partagé entre ces connexions. À l'aide de l'application de *benchmark*, nous pouvons mesurer le débit atteint par chaque paire à partir du paramètre *g* de pLogP, qui correspond au temps nécessaire pour envoyer le message à travers le lien ; en effet, le *gap* correspond à l'inverse du débit. Avec la saturation du réseau, le débit maximal atteint par chaque paire est limité, voir réduit ; par conséquent, le *gap* reflète ce partage du débit.

Pour illustrer cet expérience, nous avons mesuré le gap d'un message de 32 Mo entre les grappes d'Orsay (GdX) et Toulouse. Ainsi, la Figure 4.9 présente les valeurs mesurées et le gap moyen selon le nombre de paires connectées. Malgré la fenêtre de communication TCP adéquate à la communication *inter-grappes* (4 Mo), nous pouvons observer que l'augmentation du nombre de processus interconnectés a encore un effet direct sur le *gap*. Nous observons aussi que certaines connexions nécessitent un temps plus important pour finir la transmission de données. Ces retards indiquent l'occurrence de pertes de paquets, une caractéristique des réseaux saturés.

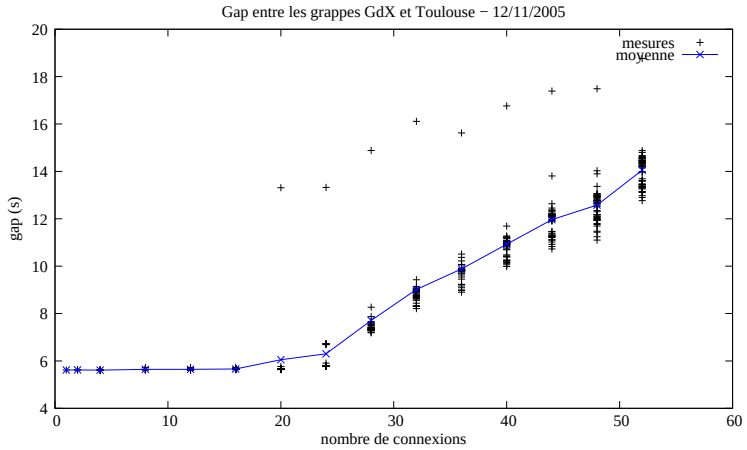


Figure 4.9 Gap *mesuré entre les grappes GdX et Toulouse*

Comme le gap correspond à l'inverse du débit (le gap étant le temps nécessaire à l'envoi d'un

message), nous pouvons obtenir facilement le débit moyen de chaque connexion à partir des valeurs de la figure précédente. Ainsi, la Figure 4.10 illustre clairement que le débit maximale atteint par chaque connexion est influencé par le nombre de connexions simultanées entre les deux grappes.

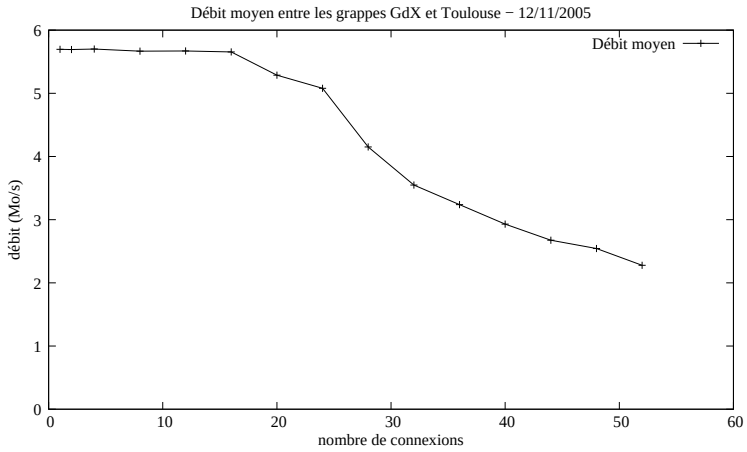


Figure 4.10 *Débit moyen par connexion entre les grappes GdX et Toulouse*

Alors que ces valeurs indiquent que le partage du débit du réseau dépend du nombre de connexions simultanées, ces résultats ne sont pas encore suffisants pour nous permettre d'affirmer que le lien entre les deux grappes a atteint sa limite. Pour cela, il faut analyser l'ensemble des connexions, de manière à établir le débit agrégé. À l'aide du paramètre g et du nombre de processus, nous pouvons obtenir le débit agrégé avec la formule suivante :

$$debit_{agrege} = \frac{m}{g(m)} \times \frac{P}{2}$$

Ainsi, le débit agrégé entre les grappes GdX et Toulouse est représenté dans la Figure 4.11. Il atteint un maximum de 120 Mo/s avec 24 connexions simultanées. À partir de ce point, le lien est saturé et l'augmentation du nombre de connexions simultanées ne permet pas l'augmentation du débit agrégé.

Alors que ce débit ne peut pas être considéré comme mauvais, il reste encore en dessous de la limite théorique du lien VTHD utilisé pour interconnecter ces deux sites. Surtout, il montre que le débit atteint par une seule paire de processus ne dépasse pas les 10 Mo/s, malgré le fait que l'environnement où ces expériences ont été réalisées était configuré avec une fenêtre TCP de 4 Mo, suffisamment grand pour atteindre le débit maximal du réseau comme indiqué par le comité de gestion de GRID5000 [GRI05]. Plusieurs facteurs peuvent influencer ce débit, comme par exemple le surcoût dû à la couche MPI. Ainsi, des expériences supplémentaires pour identifier les causes précises de ce seuil de performance devront être conduites dans le futur.

Néanmoins, l'importance de cette expérience est de mettre en évidence le fait que le réseau d'interconnexion de longue distance peut facilement être saturé par des applications à fort échange de messages. Cette constatation renforce la nécessité du développement d'algorithmes de communication adaptés aux environnements de type grille de calcul, qui sont notablement exposés à ces genres de limitations du réseau d'interconnexion.

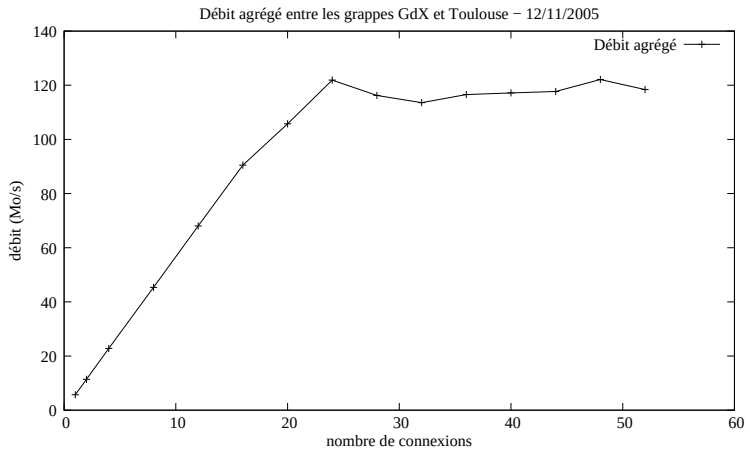


Figure 4.11 *Débit agrégé entre les grappes GdX et Toulouse*

4.9 Discussion

Alors que notre travail s'insère dans le contexte de la prédiction de performance des communications, il est tout à fait naturel que les modèles de communication soient décrits de la manière la plus précise et réaliste. Parmi autant de modèles de performance populaires, nous avons dû choisir celui qui est le plus adapté aux besoins de notre travail, notamment dans ce qui concerne la précision des prédictions et la simplicité des modèles.

Nous avons notamment pris en considération les aspects liés à la représentation des coûts de communication avec des tailles de message arbitraires, soit en communications point-à-point, soit en communications vers différents processus successivement. Notre choix sur le modèle de performance pLogP est donc lié à sa complétude et à sa précision, comme atteste Pjesivac-Grbovic *et al.* [PJE05].

Dans ce chapitre, nous avons surtout adressé les aspects liés à la communication point-à-point, notamment avec une expérience destinée à mesurer le débit effectif d'une liaison de longue distance entre deux grappes de calcul. À partir du prochain chapitre, nous allons discuter les aspects liés à la modélisation de performance des opérations de communication collective et comment intégrer ces modèles pour optimiser les communications collectives sur des grilles de calcul.

5

Les Patrons de Communication Collective

Alors que la communication entre deux processus est une caractéristique essentielle aux applications parallèles à mémoire distribuée, certaines opérations requièrent l'interconnexion entre plusieurs interlocuteurs à la fois. Si la plupart de ces opérations ont pour objectif la diffusion ou la collecte de données, d'autres fonctions comme la synchronisation des processus peuvent aussi bénéficier des communications collectives.

C'est alors tout naturellement que les bibliothèques de programmation parallèle, telles que MPI [SNI96] ou PVM offrent des primitives de communication collective. Non seulement l'utilisation de primitives spécifiquement implantées est plus commode, mais surtout cela permet aux optimisations de performance d'être pérennisées et utilisées directement par les programmeurs.

Étant donné que les opérations de communication collective permettent différents patrons de communication entre les interlocuteurs, il est important de définir une classification des patrons de communication selon le flux de données et la relation entre les processus. En effet, le flux de données peut se faire entre un ou plusieurs processus et ces données peuvent être identiques ou spécifiques à chaque interlocuteur. Pour mieux identifier ces différents flux de données, nous utilisons une nomenclature à trois paramètres : le nombre de processus sources, le nombre de processus destinataires et la nature des données. C'est ainsi, par exemple, que la diffusion d'une donnée identique à partir d'un seul processus *racine* est appelé *Un vers Plusieurs*, alors que la collecte de différentes données de plusieurs processus vers un seul processus *racine* est appelé *Plusieurs vers Un Personnalisé*.

Ces différents flux de données sont présentés dans les Tableaux 5.1 et 5.2, avec les opérations correspondantes implantées par la bibliothèque MPI. Il faut noter que MPI fait également la distinction selon la taille des messages échangés, qui peut être la même pour tous les processus ou varier selon la source ou destination du message. Nous considérons que cette différence est seulement importante pour l'implantation des opérations de communication collective et n'a pas d'influence sur la description des patrons de communication.

Dans les prochaines sections, nous présentons les différents patrons de communication collective, en faisant notamment la relation avec les opérations implantées sur MPI.

TAB. 5.1 *Patrons de Communication - message de même taille*

	Un	**Un Personnalisé**	**Plusieurs**	**Plusieurs Personnalisé**
Un vers	-	-	Broadcast	Scatter
Plusieurs vers	Barrier,Reduce	Gather	AllReduce, AllGather	Scan, AlltoAll

TAB. 5.2 *Patrons de Communication - message de taille variable*

	Un	**Un Personnalisé**	**Plusieurs**	**Plusieurs Personnalisé**
Un vers	-	-	-	Scatterv
Plusieurs vers	-	Gatherv	AllGatherv	Reduce-Scatter, AlltoAllv

5.1 Un vers Plusieurs

Dans le patron de communication *Un vers Plusieurs*, un seul processus, appelé *racine,* envoie le même message de taille m à tous les autres $(P-1)$ processus. Ce patron de diffusion, connu aussi sous le nom de *Broadcast*, est un des plus simples et répandus comme élément de base d'autres patrons de communication plus complexes.

Étant cette opération généralement unidirectionnelle (en général, le flux de données ne revient pas à un noeud qui a déjà le message), l'occurrence de congestion des ressources (sur la mémoire et le réseau) est assez faible. Par conséquent, la complexité de ce patron de communication concerne surtout la recherche des stratégies de diffusion qui minimisent le temps nécessaire à la diffusion du message entre tous les processus.

Pour cette raison, des implantations classiques du patron *Un vers Plusieurs* organisent généralement les communications sous la forme d'arbres de diffusion : de cette manière, les processus qui reçoivent les messages peuvent à leur tour renvoyer ces messages à d'autres processus, augmentant ainsi le nombre de communications exécutées en parallèle et réduisant le temps total de diffusion.

Parmi les arbres de diffusion classiques (nous appelons « classiques » les arbres de diffusion qui ne sont pas créés dynamiquement pour l'application) nous trouvons des arbres plats, binaires, binomiaux et des chaînes, comme indiqué dans la Figure 5.1. De tels arbres sont décrits par deux paramètres, d et h, où d est le nombre maximum de successeurs qu'un noeud peut avoir et h est la hauteur de cet arbre, le chemin le plus long qui relie la racine et les feuilles de cet arbre.

Plus généralement, des arbres de diffusion avec différents degrés d et h peuvent être générés à partir d'un algorithme de type *arbre-alpha* (*alpha-tree* en anglais), suggéré par Bernaschi et Ianello [BER98].

En effet, un *arbre-alpha* est un arbre dont le noeud i possède au maximum $n \times \alpha^i$ sous-branches. Dans un *arbre-alpha*, les noeuds des sous-branches sont divisés en deux groupes, S' et S" de taille n' = n - n" et n" = *round*($\alpha \times n$). Pour cela, la fonction *round* est définie comme suit :

$$round(x) = \begin{cases} 1 & if \ \lfloor x \rfloor = 0 \\ \lfloor x \rfloor & if \ \lfloor x \rfloor \neq 0 \end{cases}$$

Cette procédure donne un arbre qui est dépendant de la valeur de α, d'où le nom, a*rbre-alpha*. Un *arbre-alpha* avec $\alpha = 0.5$ est équivalent à un arbre binomial ; des valeurs de α suffisamment petites donnent des arbres plats. Pour la construction dynamique des *arbres-alpha*, Bernaschi *et al.* [BER98] ont suggéré le pseudo-code présenté par l'Algorithme 2.

Ces mêmes auteurs ont observé que, pour la plupart des systèmes homogènes, les arbres bino-

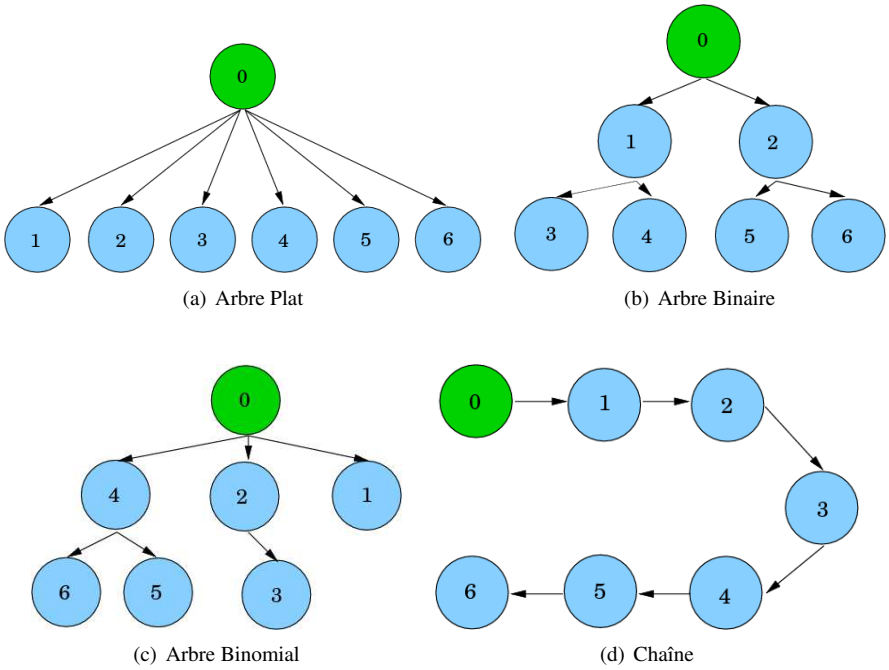


Figure 5.1 *Exemples d'arbres de diffusion*

miaux sont optimaux et, en conséquence, la majorité des implantations MPI l'utilise comme implantation par défaut de l'opération *Broadcast*.

Algorithme 2 Pseudo-code pour la construction d'un *arbre-alpha*

```
this ← myID
parent ← 0
n ← P
while this != 0 do
    nOther ← round(α × n)
    threshold ← n - nOther
    if this = threshold then
        n ← nOther
        this ← this - threshold
    else if this < threshold then
        n ← threshold
    else
        n ← nOther
        parent ← parent + threshold
        this ← this - threshold
    end if
end while
children ← <>
while n > 1 do
    n ← n - round(α × n)
    children += children + <myID + n>
end while
```

Dans la bibliothèque de communication MPI [SNI96], l'opération de communication collective qui fournit le patron de communication *Un vers Plusieurs* est appelé *MPI_Bcast*, dont la définition

est la suivante :

```
MPI_Bcast(buffer, count, datatype, root, comm)
    INOUT buffer starting address of buffer
    IN count number of entries in buffer
    IN datatype data type of buffer
    IN root rank of broadcast
    IN comm communicator
```

L'opération MPI_Bcast diffuse un message d'un processus *racine* (identifié par le paramètre *root*) vers tous les autres processus du groupe. Le paramètre *root* doit avoir la même valeur dans tous les processus, ainsi que le paramètre *comm*, qui représente le groupe de communication. Au retour de l'opération *MPI_Bcast*, le contenu du buffer de communication du processus racine est copié dans tous les processus, comme illustré dans la Figure 5.2.

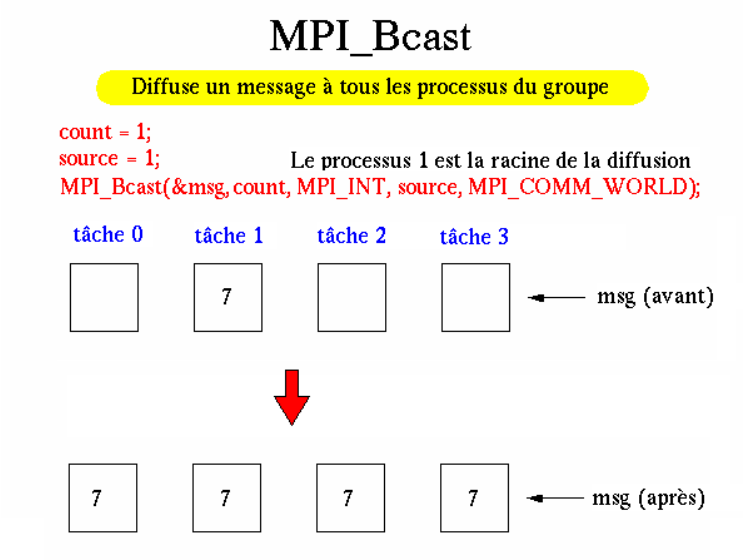


Figure 5.2 *Fonctionnement de l'opération* MPI_Bcast

Par exemple, l'Algorithme 3 illustre un cas où les processus doivent chercher l'occurrence d'un numéro (appelé *cible*) dans une liste de données. Pour cela, le processus maître peut utiliser l'opération *MPI_Bcast* pour diffuser cette valeur cible à tous les processus.

Algorithme 3 Exemple d'utilisation de MPI_Bcast

```
MPI_Comm comm;
int cible;
int root=0;
...
MPI_Bcast ( cible, 1, MPI_INT, root, comm);
...
```

Parmi les travaux qui visent l'optimisation de l'opération *MPI_Bcast* pour les environnements homogènes, nous pouvons citer ceux de Karp [KAP93], Van de Geijn [BAT96], Kesavan [KES97], Huse [HUS99], Thakur [THA03], Juhasz [JUH04], Träff [TRA04] et Vadhiyar [VAD00][VAD04].

Le travail de Karp [KAP93], orienté vers la modélisation des algorithmes de broadcast en utilisant le modèle de coût LogP, a soulevé certaines questions sur la modélisation avec différents

régimes de travail. Il a étudié notamment la diffusion d'un seul item de données, de k items de données et la diffusion en régime permanent. Ce dernier cas, appelé *steady state* en anglais, a été repris dans les travaux de Beaumont et *al.* [BEA04a, BEA04b, BEA05a, BEA05b] dans le cas des réseaux hétérogènes.

Le concept de diffusion de k items a été ensuite développé par Van de Geijn *et al.* [BAT96], qui ont étudié la segmentation des messages dans le cadre des opérations de type *Broadcast*, dont les chaînes segmentées (*pipelines*) et l'algorithme *Scatter/Collect* (*Scatter* suivi de *Allgather*). Dans ce même sens, le travail de Kesavan [KES97] a considéré la segmentation des messages, cette fois-ci dans le cadre de l'implantation de primitives de communication de bas niveau (interface réseau) spécifiques à la diffusion des messages.

Toutefois, il reste le fait que les performances des différentes stratégies d'implantation dépendent des performances du réseau, qui varient selon la latence, le débit et la taille des messages. Bruck [BRU96] a suggéré l'utilisation du modèle de coût Postal afin de construire des arbres de diffusion adaptés aux paramètres de performance du réseau. Cependant, la construction dynamique des arbres de communication pour un environnement homogène peut s'avérer trop coûteuse par rapport aux gains réels. Ainsi, les travaux de Huse [HUS99] et Vadhiyar [VAD00][VAD04] ont préféré comparer différentes stratégies « classiques » à partir d'expériences pratiques, en vue d'identifier la stratégie la plus adaptée. Dans le même sens, Nishtala *et al.* [NIS03] ont suggéré une approche pratique dont un algorithme d'analyse statistique utilise une base de données qui contient les performances des implantations et la trace des variations de performance du réseau, afin d'identifier l'implantation la plus optimisée pour une situation déterminée.

Toutefois, si les expériences de Vadhiyar *et al.* donnent des estimations très fiables de la performance des opérations de communication collective, le coût initial de cette approche n'est pas négligeable. En effet, il faut comparer les résultats pour un nombre important de paramètres (taille du message, nombre de noeuds, etc.) et de stratégies d'implantation. C'est en raison de ce coût élevé que le travail plus récent de Vadhiyar ([VAD04]) suggère l'utilisation conjointe des expériences pratiques et des prédictions de performance obtenues avec des modèles de communication. En effet, notre travail a démontré que la modélisation des performances est suffisamment précise pour permettre l'identification des stratégies de communication les plus adaptées à un environnement de calcul donné [BAR04b, BAR04c, BAR05a].

Plus récemment, des travaux comme ceux de Thakur [THA03], Juhasz [JUH04] et Träff [TRA04] ont repris les concepts de Van de Geijn, qui implante le *Broadcast* à partir d'une opération *Scatter* suivie d'une opération *Allgather*. Thakur, Juhasz et Träff ont notamment amélioré la performance de l'opération *Allgather*, en utilisant différents algorithmes comme l'anneau ou le *recursive doubling* ou en parallélisant les étapes de diffusion/collecte des segments, comme dans l'algorithme de Träff. En effet, le travail de Chan [CHA04] a mené une profonde recherche sur les combinaisons de différents patrons de communication, à l'exemple de la stratégie employée par Van de Geijn.

Si ces travaux visent l'optimisation des opérations de *broadcast* dans les environnements homogènes, ils existent aussi plusieurs auteurs qui se sont intéressés aux systèmes hétérogènes. C'est le cas des grappes hétérogènes, comme dans les travaux de Banikazemi [BAN98], Bhat [BHA03], Figueira[FIG04] et Liu [Liu00b], les grappes de machines à multiprocesseurs (SMP), comme dans les travaux de Sun [SUN02] et Wu [WUM04], ou les grilles de calcul, comme par exemple, les travaux de Karonis [KAR00, KAR02] et Kielmann [KIE99b, KIE01]. Nous présenterons tous ces travaux en détails dans le chapitre 7, qui est consacré à l'optimisation des communications collectives dans les environnements hétérogènes, notamment les grilles de calcul.

5.2 Un vers Plusieurs Personnalisé

Dans le patron de communication *Un vers Plusieurs Personnalisé*, le processus *racine* détient P messages différents de taille m qui seront distribués également entre tous les P processus.

À l'opposé du patron *Un vers Plusieurs*, chaque processus reçoit un message différent, comme par exemple, des tâches de travail ou des tranches de données. En effet, comme chaque message est adressé à un seul processus, les implantations de ce patron de communication généralement utilisent une stratégie en arbre plat, où le processus *racine* communique directement avec chacun des processus. Néanmoins, cela n'exclut pas l'utilisation d'autres stratégies de communication, comme indiquent Alexandrov [ALE95], Huse [HUS99] et Vadhiyar [VAD00, VAD04]. Nous avons même montré que certaines techniques ont des performances très avantageuses et ceci pour des réseaux très différents comme Fast Ethernet, Giga Ethernet et Myrinet [BAR04a, BAR04c, BAR05a]. Cependant, leur efficacité est très liée aux caractéristiques du réseau, raison par laquelle les implantations de MPI utilisent la stratégie en arbre plat, plus générale.

Dans la bibliothèque de communication MPI [SNI96], l'opération de communication collective qui fournit le patron de communication *Un vers Plusieurs Personnalisé* est appelée *MPI_Scatter*, dont la définition est la suivante :

```
MPI_Scatter(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)
    IN sendbuf address of send buffer
    IN sendcount number of elements send to each process
    IN sendtype datatype of send buffer elements
    OUT recvbuf address of receive buffer
    IN recvcount number of elements in receive buffer
    IN recvtype data type of recv buffer elements
    IN root rank of sending process
    IN comm communicator
```

L'opération *MPI_Scatter* agit comme si le processus racine exécutait *n* opérations d'envoi de type *MPI_Send(sendbuf+i-sendcount-extent (sendtype), sendcount, sendtype, i,...)*, dont *i* varie entre *0* et *n - 1*, alors que chaque autre processus exécute une opération de réception de type *MPI_Recv (recvbuf, recvcount, recvtype, root,...)*.

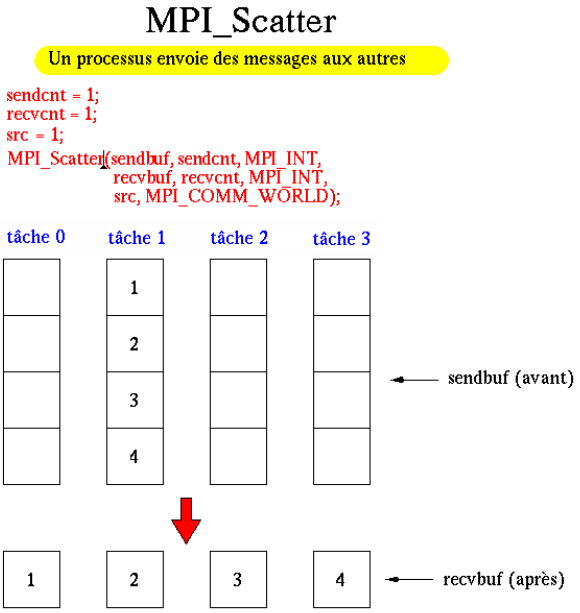


Figure 5.3 *Fonctionnement de l'opération* MPI_Scatter

Une autre alternative pour décrire l'opération *MPI_Scatter* est que le processus racine envoie un message de taille *n.* Ce message est segmenté en *n* messages de même taille, dont le segment *i* est envoyé au processus *i* du groupe, comme illustré par la Figure 5.3.

Pour démontrer l'utilisation de *MPI_Scatter*, nous reprenons l'exemple de la section précédente (*MPI_Bcast*). Maintenant, le processus maître, en plus de diffuser la valeur cible, donne à chaque processus une partie de la liste de numéros pour faire la recherche de *cible*. Le processus maître peut utiliser l'opération *MPI_Scatter* pour répartir la liste de numéros entre tous les processus, comme illustré dans l'Algorithme 4.

Algorithme 4 Exemple d'utilisation de *MPI_Scatter*

```
MPI_Comm comm;
int rank, gsize;
int *sendarray, *rbuf;
...
MPI_Comm_rank(comm, &rank);
MPI_Comm_size(comm, &gsize);
rbuf=(int *) malloc (100*sizeof(int));
if (rank == 0)
    sendarray=(int *) malloc (gsize*100*sizeof(int));
MPI_Scatter ( sendarray, 100, MPI_INT, rbuf, 100, MPI_INT, 0, comm);
...
```

Parmi les rares travaux qui suggèrent des optimisations réelles pour l'algorithme de *Scatter* nous pouvons citer Alexandrov [ALE95] et Bruck [BRU97a]. Ces deux auteurs suggèrent l'utilisation de différents algorithmes selon la taille du message. En effet, Alexandrov et Bruck suggèrent que pour des petits messages, il est plus efficace de diffuser (en utilisant un *Broadcast*) tous les messages dans un seul « bloc », car le coût de l'envoi de plusieurs petits messages est généralement plus élevé que la diffusion d'un message unique.

Dans le cas de grands messages, Alexandrov a analysé la performance des implantations en arbre plat et binomial. Il a constaté que, malgré la meilleure performance des arbres binomiaux, ceux-ci sont parfois sous-optimaux, car leur structure ne permet pas un bon équilibrage des transmissions et des réceptions. Son travail présente donc un algorithme optimal pour la construction d'arbres de diffusion *Scatter* quand la latence est plus grande que le coût d'envoi (L > g) ; ce même algorithme revient automatiquement au format d'arbres binomial si L <= g. Malheureusement, l'évolution des réseaux locaux a réduit considérablement la latence d'interconnexion, de manière à ce que l'algorithme d'Alexandrov n'est vraiment utile qu'avec des réseaux de longue distance, qui sont mieux optimisés par d'autres travaux.

D'une manière générale, la plupart des travaux sur le *Scatter* visent soit la modélisation des performances, comme les travaux de Banikazemi [BAN98] Touriño [TOU99] et Chan [CHA04], soit l'optimisation pour les environnements hétérogènes, comme les travaux de Miled [MIL98] et de Kielmann [KIE99b, KIE01]. Dans le cas des environnements hétérogènes, la problématique n'est plus l'élaboration des stratégies d'implantation mais la réorganisation des communications, afin de réduire le temps total d'exécution.

5.3 Plusieurs vers Un

Dans le patron de communication *Plusieurs vers Un*, un processus racine reçoit le même message de taille m envoyé par les $P-1$ autres processus. Comme tous les messages reçus sont identiques, ce patron de communication a une utilisation plus restreinte, à moins qu'il ne soit utilisé en association à d'autres patrons de communication.

Un des rares exemples où le patron *Plusieurs vers Un* est employé directement est celui des protocoles de réplication active [DEF00]. Dans ce cas-là, différentes répliques d'un processus exécutent

la requête du client simultanément et envoient la réponse (censé être la même) aussi tôt que la requête est exécutée. Le client, qu'ici a le rôle de *racine*, attend jusqu'à l'arrivée de la réponse la plus rapide et rejette (ignore) les autres réponses.

5.3.1 MPI_Barrier

Clairement, l'utilisation du patron *Plusieurs vers Un* présenté en haut n'est pas triviale dans le domaine du calcul parallèle. En revanche, ce patron est fréquemment utilisé en association avec le patron *Un vers Plusieurs* : l'opération *MPI_Barrier* les utilise pour synchroniser les processus, en établissant une barrière de communication où tous les processus doivent être présents pour que la tâche continue, comme illustré par Figure 5.4.

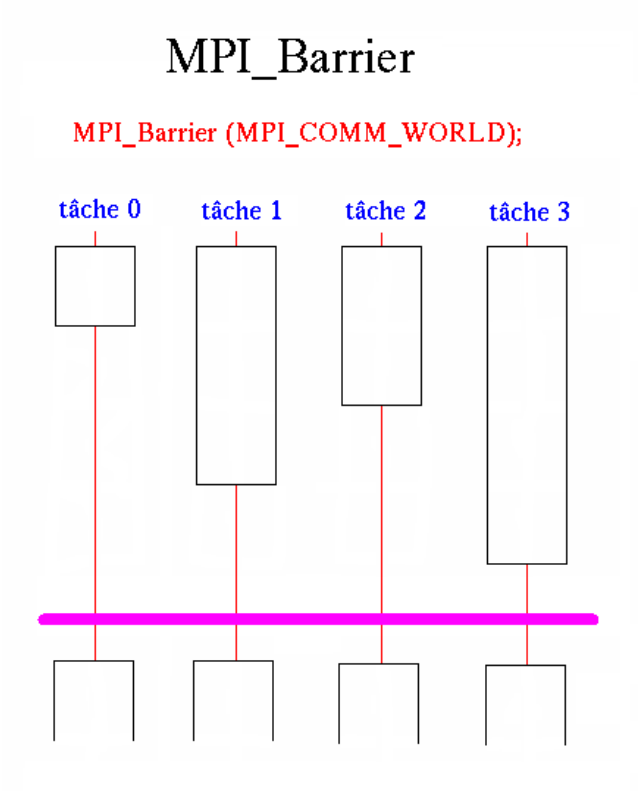


Figure 5.4 *Fonctionnement de l'opération* MPI_Barrier

Pour synchroniser les processus, les implantations MPI généralement utilisent une combinaison d'autres opérations de communication collective. Ainsi, une implantation classique de *MPI_Barrier* exécute d'abord le patron de communication *Plusieurs vers Un* : chaque processus envoie au processus *racine* un message (normalement avec une taille proche de zéro) pour indiquer qu'il est arrivé à la barrière. Une fois que le processus *racine* a reçu les messages de tous le autres processus, il exécute une opération de type *Un vers Plusieurs* (*Broadcast*) pour annoncer que tous les processus sont désormais synchronisés et qu'ils peuvent continuer la tâche.

Ainsi, selon [SNI96], la définition de l'opération *MPI_Barrier* est la suivante :

```
MPI_Barrier(comm)
```

```
IN comm communicator
```

Comme le patron *Plusieurs vers Un* représente l'opération contraire du patron *Un vers Plusieurs* par rapport au flux de données, son implantation peut bénéficier des mêmes stratégies de communication de celui-là, notamment les stratégies en arbre binomial.

Parmi les optimisations les plus courantes, Vadhiyar *et al.* [VAD00] et Pjesivac-Grbovic *et al.* [PJE05] utilisent différentes stratégies pour l'implantation d'une barrière, utilisant notamment les algorithmes *double anneau*, Bruck [BRU97b], *recursive doubling* [THA03] et *tournoi* [HEN88].

Dans l'algorithme *double anneau*, qui nécessite $O(P)$ étapes de communication, un message de taille zéro est envoyé à partir d'un processus *racine* vers le processus suivant et ainsi de suite. Par conséquent, un processus ne peut continuer que s'il a reçu le message deux fois (l'une pour synchroniser les processus, l'autre pour le signaler).

Pour l'algorithme de Bruck, $\lceil log_2 P \rceil$ étapes de communication sont nécessaires. À l'étape k, le processus r reçoit un message de taille zéro octets du processus de rang $(r - 2^k)$ et envoie vers le processus de rang $(r + 2^k)$.

L'algorithme *recursive doubling* nécessite $log_2 P$ étapes de communication si P est une puissance de 2 et $\lfloor log_2 P \rfloor + 2$ étapes dans le cas contraire. Ainsi, à l'étape k, le processus r échange des messages avec le processus (r XOR 2^k), comme illustré par Figure 5.5. Si le nombre de processus P n'est pas une puissance de 2, deux autres étapes de communication sont nécessaires pour traiter les processus supplémentaires.

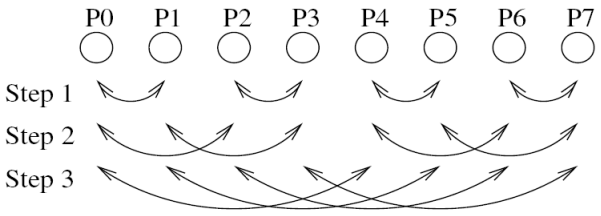


Figure 5.5 *Structure de communication pour un algorithme* recursive-doubling

Finalement, l'algorithme du *tournoi* nécessite aussi $\lceil log_2 P \rceil$ étapes de communication. Dans cet algorithme, similaire à l'approche en arbre binomial, la barrière est vue comme un tournoi entre les processus, où les processus qui reçoivent des messages avancent à la prochaine étape du tournoi. Quand finalement le « champion » est défini, celui-ci envoie un message vers tous les processus (*Broadcast*).

Cependant, une caractéristique de l'opération *MPI_Barrier* est qu'elle ne comporte pas de transmission de données, seulement la coordination entre les processus. De ce fait, certains systèmes utilisent des primitives de bas niveau pour gérer efficacement cette opération. C'est le cas de Petrini *et al.* [PET03], dont l'architecture réseau permet le traitement direct de certains messages par l'interface réseau et l'envoi de messages de type multicast. Dans leur système, les processus qui arrivent à la barrière écrivent en mémoire un numéro de séquence qui les identifie. Le processus *racine* envoie régulièrement un message à chaque processus pour vérifier si tous sont synchronisés ; cependant, ces messages sont traités directement par l'interface réseau, qui en plus d'envoyer le statut de l'adresse mémoire, combine les diverses réponses de manière à ce que le processus racine reçoive un seul message. Dés que le processus *racine* détecte que tous les processus sont synchronisés, il envoie un message *multicast* en utilisant les primitives du matériel du réseau, débloquant ainsi tous les processus avec un seul message.

5.3.2 MPI_Reduce

Légère variation du patron *Plusieurs vers Un*, l'opération *MPI_Reduce* permet le regroupement des différentes données des processus par le biais des opérations mathématiques ou logiques.

Plus exactement, l'opération *MPI_Reduce* combine les éléments contenus dans les buffers d'envoi de chaque processus du groupe en utilisant l'opération *op*, dont la valeur résultante est déposée dans le buffer de sortie du processus *racine*, comme illustré par Figure 5.6. Les buffers d'entrée et de sortie sont définis par le type de données et leur taille, qui doivent être les mêmes pour les deux buffers. Aussi, les paramètres o*p*, *root* et *comm* doivent être identiques dans tous les processus. Dans le cas où les processus fourniraient des séquences d'éléments, les opérations sont exécutées élément par élément. Par exemple, si l'opération MPI_MAX est exécutée sur un buffer qui contient deux numéros réels *(count = 2 et datatype = MPI_FLOAT)*, alors *recvbuf[0] = globalmax(sendbuf[0])* et *recvbuf[1] = globalmax(sendbuf[1])*.

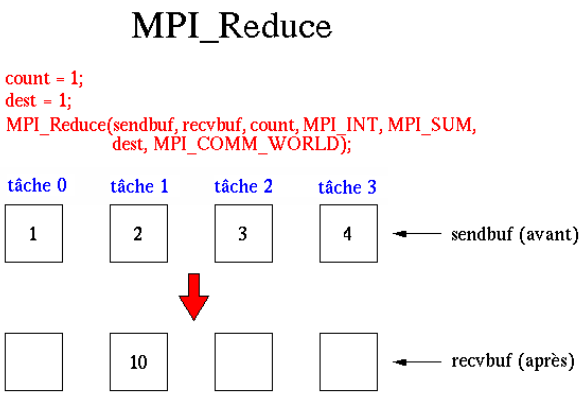


Figure 5.6 *Fonctionnement de l'opération* MPI_Reduce

La définition de l'opération *MPI_Reduce* est la suivante [SNI96] :

```
MPI_Reduce(sendbuf, recvbuf, count, datatype, op, root, comm)
    IN sendbuf address of send buffer
    OUT recvbuf address of receive buffer
    IN count number of elements in the send buffer
    IN datatype datatype of send buffer elements
    IN op reduce operation
    IN root rank of root process
    IN comm communicator
```

Pour implanter l'opération *MPI_Reduce*, l'approche la plus simple, comme indiqué par Vadhiyar *et al.* [VAD00], est l'utilisation d'une opération de type *MPI_Gather* suivie de calcul, obligeant ainsi que toutes les données soient transmises au processus *racine*.

Or, non seulement le processus *racine* doit traiter $P-1$ messages, mais il doit aussi effectuer $P-1$ opérations. Pour réduire le nombre de messages et d'opérations, des alternatives similaires à celles déjà employées pour l'opération *MPI_Bcast* peuvent être utilisées, notamment les arbres binaires ou arbres binomiaux [VAD00]. Dans ce cas, les opérations partielles de réduction se déroulent à chaque étape de communication, où chaque processus fait la réduction entre les messages reçus de ses processus « fils » et son propre message avant d'envoyer le résultat vers son processus « père ». En effet, l'approche avec des arbres binomiaux, qui peut être exécuté en seulement $\lceil \log_2 P \rceil$ étapes de communication, est largement utilisée, dont pour l'implantation LAM-MPI 7.0.4 [LAM04].

Folino *et al.* [FOL98] ont évalué la performance des opérations *MPI_Reduce* construites avec des arbres binomiaux et ont observé que la performance était assez proche de celle du *MPI_Bcast* pour des petits messages ; le coût du traitement et du calcul devenait suffisamment important seulement quand la taille des messages augmentait. Touriño *et al.* [TOU99] ont détecté des différences plus significatives même pour des petits messages, ce qui s'explique en partie par le fait qu'une opération de type *MPI_Reduce* est beaucoup plus dépendante de la synchronisation des processus qu'un *MPI_Bcast*.

Néanmoins, le coût du calcul peut devenir assez important selon la taille des messages et l'opération de réduction. Ce coût relatif au calcul, γ, est utilisé par Thakur *et al.* [THA03] et Chan *et al.* [CHA04] pour augmenter leur modèle de performance. Ainsi, le coût réel d'une opération *MPI_Reduce* fondé sur un arbre binomial est approximativement $\lceil log_2 P \rceil \times L + \lfloor log_2 P \rfloor \times (g(m) + \gamma(m))$.

Si l'algorithme de réduction fondé sur un arbre binomial est très intéressant pour les petits messages, à cause du nombre réduit d'étapes de communication, il n'est pas très efficace pour des grands messages. Dans ce cas-là, l'algorithme proposé par Rabenseifner [RAB97] est plus indiqué. L'algorithme de Rabenseifner suit le principe du travail de Van de Geijn pour l'implantation de l'opération *MPI_Bcast* [BAT96]. Van de Geijn implante le *Broadcast* à partir d'une opération *Scatter* suivie d'une opération *Allgather*, alors que Rabenseifner implante l'opération de *Reduce* pour des grands messages comme une opération *Reduce-scatter* suivie d'un *Gather* vers le processus racine. Ces deux techniques ont pour résultat la réduction du coût de bande passante de $\lfloor log_2 P \rfloor \times g(m)$ vers $2 \times g(m)$. Ainsi, dans le cas de l'implantation MPICH [MPI03], la stratégie de Rabenseifner est utilisée pour des messages longs (plus de 2 Koctets), alors que l'implantation avec des arbres binomiaux est utilisée pour des messages plus petits que 2 Koctets.

Malheureusement, la performance de ces algorithmes dépend fortement de l'homogénéité des processus. En effet, la présence de processus moins performants peut influencer considérablement le temps d'exécution d'une opération de réduction. Cette observation est notamment valable pour les algorithmes de type arbre, dont l'exécution des réductions partielles dépend de la réception préalable des messages des processus « fils ». En effet, il suffit qu'un processus soit moins performant pour que le retard s'accumule et se propage dans l'arbre de réduction. Liu [LIU00a] a étudié ce problème à travers la définition d'une heuristique de construction d'arbres de réduction, *Slowest Node First*, qui réorganise les communications afin d'éviter une attente trop élevée des messages issus des processus moins performants.

Un autre point important à considérer est que, par défaut, l'opération *op* est toujours considérée comme associative et les opérations prédéfinies par MPI (listées dans le Tableau 5.3) sont aussi commutatives. Initialement, l'ordre d'évaluation des opérandes est déterminé par le rang des processus dans le groupe, mais les implantations peuvent changer cet ordre pour augmenter la performance de cette opération. Bien sur, les utilisateurs peuvent encore définir des opérations qui ne sont pas commutatives, mais les différentes implantations de *MPI_Reduce* peuvent changer le résultat de certaines opérations, comme par exemple l'addition des numéros réels.

Pour éviter cela, certaines implantations utilisent différentes implantations de MPI_Reduce selon le type d'opération. Dans le cas de MPICH [THA03], les implantations à haute performance comme celle de Rabenseifner sont utilisées exclusivement avec les opérations de réduction prédéfinies. Dans le cas des opérations fournies par l'utilisateur, seulement l'algorithme en arbre binomial est utilisé, car non seulement l'ordre des opérations est respecté, mais aussi cela évite le traitement de types dérivés, qui rendent difficile l'opération Reduce-scatter.

De même, Kielmann [KIE99a] utilise différentes stratégies selon les caractéristiques de l'opération de réduction. Cependant, il laisse à l'utilisateur le choix entre les stratégies et l'utilisateur doit indiquer explicitement si l'opération est strictement associative.

TAB. 5.3 *Opérations Reduce prédéfinies*

Nom	Description
MPI_MAX	maximum
MPI_MIN	minimum
MPI_SUM	somme
MPI_PROD	produit
MPI_AND	*ET* logique
MPI_BAND	*ET* binaire
MPI_OR	*OU* logique
MPI_BOR	*OU* binaire
MPI_XOR	*XOR* logique
MPI_BXOR	*XOR* binaire
MPI_MAXLOC	maximum et localisation
MPI_MINLOC	minimum et localisation

5.4 Plusieurs vers Un Personnalisé

Dans le patron de communication *Plusieurs vers Un Personnalisé*, un processus racine reçoit différents messages de taille m envoyés par les $P-1$ autres processus, comme illustré par Figure 5.7. Ce patron de communication est conceptuellement l'opération contraire du patron *Un vers Plusieurs Personnalisé*, et pour cela, peut bénéficier des mêmes stratégies d'implantation que celui-là.

Dans la bibliothèque de communication MPI, l'opération de communication collective qui utilise le patron de communication *Plusieurs vers Un Personnalisé* est appelée *MPI_Gather*, dont la définition est la suivante [SNI96] :

```
MPI_Gather(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)
    IN sendbuf starting address of send buffer
    IN sendcount number of elements in send buffer
    IN sendtype data type of send buffer elements
    OUT recvbuf address of receive buffer
    IN recvcount number of elements for any single receive
    IN recvtype data type of recv buffer elements
    IN root rank of receiving process
    IN comm communicator
```

Avec *MPI_Gather*, chaque processus envoie le contenu de son buffer vers le processus racine. Le processus racine les reçoit et sauvegarde ces messages selon l'ordre des rangs des processus. Une autre alternative pour décrire l'opération *MPI_Gather* est qu'un groupe de n messages envoyés par les processus sont regroupés selon le rang des processus, et le message résultant est reçu par le processus *racine* comme s'il avait exécuté *MPI_Recv(recvbuf, recvcount-n, recvtype, ...)*.

Comme l'opération *MPI_Gather* est l'opération inverse de *MPI_Scatter* par rapport au flux de données, *MPI_Gather* peut bénéficier de la plupart des optimisations de celle-là [HUS99] [VAD00, VAD04]. En effet, la modélisation des performances des deux opérations est souvent similaire [CHA04], et on peut attendre des résultats assez proches dans des environnements strictement homogènes.

Cependant, l'existence de processus moins performants que d'autres peut influencer considérablement le temps d'exécution de l'opération *Gather*, notamment si l'implantation utilise des arbres avec dépendances entre les processus. Hatta [HAT00] et Ooshita [OOS02] ont repris les concepts de Banikazemi [BAN98] et Liu [LIU00a] pour le *Broadcast* et le *Reduce*, respectivement, et ont dé-

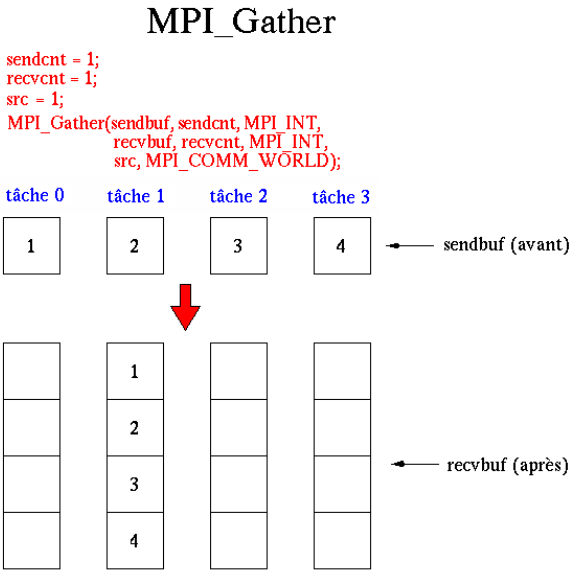


Figure 5.7 *Fonctionnement de l'opération* MPI_Gather

veloppé des heuristiques pour ordonnancer les communications de manière à compenser les retards des processus les plus lents.

5.5 Plusieurs vers Plusieurs

Les patrons de type *Plusieurs vers Plusieurs* ou *Plusieurs vers Plusieurs Personnalisé* sont caractérisés par l'absence d'un processus *racine*. Dans le cas du patron *Plusieurs vers Plusieurs*, les messages envoyés par les processus sont regroupés comme dans le cas des patrons *Plusieurs vers Un* ou *Plusieurs vers Un Personnalisé*, mais ce résultat est ensuite diffusé vers tous les processus, comme dans le patron *Un vers Plusieurs*.

De ce fait, les stratégies pour implanter ce patron de communication sont très variées. Les stratégies les plus simples appliquent des techniques bien connues pour les différentes étapes (regroupement, diffusion), alors que des solutions spécifiquement développées, comme certaines indiquées par [VAD00] peuvent être bien plus efficaces.

Dans la bibliothèque de communication MPI, deux opérations utilisent le patron de communication *Plusieurs vers Plusieurs*, *MPI_Allreduce* et *MPI_Allgather*.

5.5.1 MPI_Allreduce

La première opération, appelé *MPI_Allreduce*, distribue à tous les processus le résultat de la réduction des données contenues dans chaque processus, comme illustré par Figure 5.8. Pour cela, tous les processus doivent obligatoirement recevoir des résultats identiques.

Par conséquent, une implantation de *MPI_Allreduce* pourrait se faire avec un appel à *MPI_Reduce* suivi d'un appel à *MPI_Bcast* ou alors un appel à *MPI_Allgather* suivi d'un appel à *MPI_Scatter* [VAD00]. En effet, plusieurs combinaisons sont possibles, comme démontre Chan *et al.* [CHA04],

dont l'algorithme proposé par Rabenseifner [RAB97], qui est composé d'un *Reduce-scatter* suivi d'un *Allgather*.

Tan [TAN02] a aussi contribué avec l'évaluation de plusieurs stratégies d'implantation, dont on peut citer le *recursive-halving/recursive-doubling*, *fan-in/fan-out*, *full fan-in*, *échange bidirectionnel* et *anneau.* Ses observations ont démontré que la taille des messages est très importante pour déterminer le meilleur algorithme. En effet, Thakur *et al.* [THA05], utilisent l'algorithme de Rabenseifner pour les grands messages, alors que pour les petits messages ils utilisent un algorithme de type *recursive doubling* avec des opérations de réduction à chaque étape de communication.

```
MPI_Allreduce(sendbuf, recvbuf, count, datatype, op, comm)
    IN sendbuf address of send buffer
    OUT recvbuf address of receive buffer
    IN count number of elements in the send buffer
    IN datatype datatype of send buffer elements
    IN op reduce operation
    IN comm communicator
```

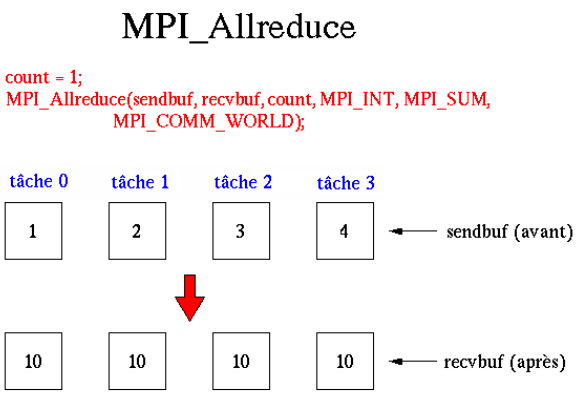


Figure 5.8 *Fonctionnement de l'opération* MPI_Allreduce

5.5.2 MPI_Allgather

La deuxième opération qui utilise le patron *Plusieurs vers Plusieurs*, *MPI_Allgather*, diffuse à tous les processus l'ensemble de données envoyées par chaque processus, comme illustré par la Figure 5.9. *MPI_Allgather* est similaire à l'opération *MPI_Gathe*r, à l'exception que tous les processus reçoivent le résultat. Ainsi, le bloc de données envoyé par le processus de rang *j* est enregistré à la position *j* du buffer recvbuf de chaque processus.

```
MPI_Allgather(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)
    IN sendbuf starting address of send buffer
    IN sendcount number of elements in send buffer
    IN sendtype data type of send buffer elements
    OUT recvbuf address of receive buffer
    IN recvcount number of elements for any single receive
    IN recvtype data type of recv buffer elements
    IN comm communicator
```

Plusieurs stratégies peuvent être utilisées pour implanter l'opération *MPI_Allgather*. Si l'approche la plus simple est l'utilisation conjointe d'un *Gather* suivi par un *Broadcast*, comme rappelle

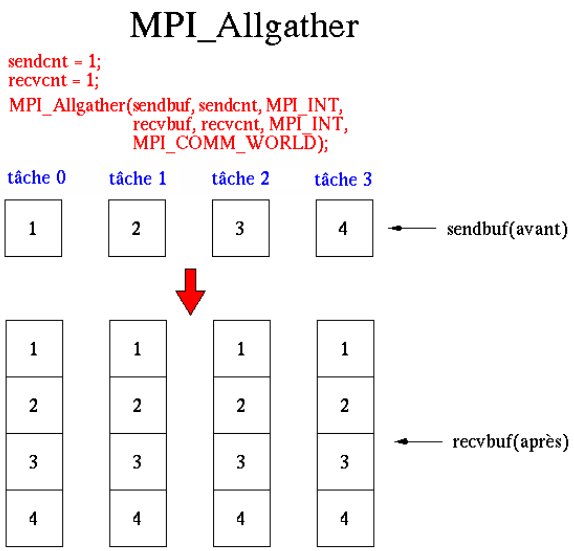


Figure 5.9 *Fonctionnement de l'opération* MPI_Allgather

Vadhiyar *et al.* [VAD00], ou alors un *anneau* virtuel [THA03], le nombre d'étapes de communication reste élevé, ce qui réduit considérablement la performance de ces implantations, comme constate Karp [KAP93].

Des algorithmes plus efficaces furent étudiés par Benson *et al.* [BEN03]. Le premier de ces algorithmes, le *recursive doubling*, également étudié par Chen [CHE96], a été utilisé par certaines implantations de MPICH [THA03]. Il est assez efficace pour les opérations dont le nombre de processus est une puissance de 2, car à chaque étape, les processus à une distance $2^k, (0 \leq k \leq \lceil \log_2 P \rceil)$ échangent leurs données. Quand le nombre de processus n'est pas une puissance de 2, $\lceil \log_2 P \rceil$ étapes de communication additionnelles sont nécessaires pour collecter les données des processus supplémentaires.

Le deuxième algorithme pour l'*Allgather* est celui de Bruck [BRU97b], qui nécessite seulement $\log_2 P$ étapes de communication même pour un nombre de processus différent d'une puissance de 2. Dans cet algorithme, à chaque étape de communication k, $(0 \leq k \leq \lceil \log_2 P \rceil)$, le processus i envoi les données qu'il contient vers les processus $(i - 2^k)$ et reçoit des données du processus $(i + 2^k)$. Cet algorithme est terminé par une copie locale en mémoire, nécessaire pour réordonner les messages reçus.

Thakur *et al.* [THA05] ont évalué ces deux algorithmes pour les utiliser dans l'implantation MPICH 1.2.6. Selon leur observation, les algorithmes de Bruck et de *recursive doubling* sont plus efficaces pour des messages petits, alors que l'algorithme en *anneau* est plus adapté aux messages longs. Ils ont observé aussi que l'algorithme de Bruck est légèrement moins efficace pour les opérations dont le nombre de processus est une puissance de 2 car il nécessite l'étape de permutation des messages, alors que le *recursive doubling* n'en a pas besoin. Par conséquent, l'implantation de MPICH utilise plusieurs algorithmes, dont le plus performant est utilisé selon la taille des messages et le nombre de processus.

À ces résultats s'ajoutent ceux de Tan [TAN02], qui a évalué plusieurs stratégies d'implantation, dont l'algorithme de *recursive-doubling*, et les travaux de Jacunski *et al.* [JAC99] et Han *et al.* [HAN00], qui ont comparé les performances de certaines implantations dans le cadre des réseaux

composés de multiples commutateurs.

5.6 Plusieurs vers Plusieurs Personnalisé

5.6.1 MPI_Alltoall

Un des plus importants patrons de communication collective pour des applications scientifiques est le *Plusieurs vers Plusieurs Personnalisé* ou échange total [CHR99], où les algorithmes parallèles échangent des données entre tous les processus. Pour cela, une des opérations le plus répandue est le *MPI_Alltoall*, qui permet la transposition des données appartenant à un groupe de processus, comme illustré par la Figure 5.10.

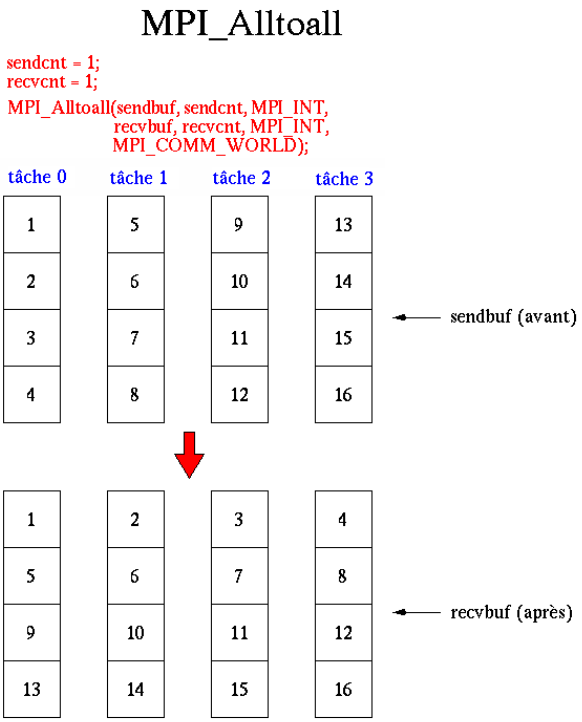


Figure 5.10 *Fonctionnement de l'opération* MPI_Alltoall

Dans le cas de l'opération *MPI_Alltoall*, chaque processus détient $m \times P$ unités de données qui seront distribuées également entre les P processus ; la syntaxe de l'opération est présentée en bas. Ce patron de communication est fréquemment utilisé pour des transpositions de matrice, transformations de Fourier bidimensionnelles, conversion entre des schémas de stockage de données (par exemple, la conversion des matrices entre applications C et Fortran), permutations et la multiplication de matrices-vecteurs.

```
MPI_Alltoall(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)
    IN sendbuf starting address of send buffer
    IN sendcount number of elements in send buffer
    IN sendtype data type of send buffer elements
    OUT recvbuf address of receive buffer
```

```
IN recvcount number of elements for any single receive
IN recvtype data type of recv buffer elements
IN comm communicator
```

Historiquement, des techniques spécifiques pour des environnements réseaux spéciaux ont été largement recherchées, comme par exemple, les réseaux disposés en *tores* [CAL95], *grilles*, *hypercubes* [KAL03] et aussi les réseaux d'interconnexion de type *omega* [YAN00]. Cependant, la diversité des systèmes actuelle requiert des solutions plus générales. En conséquence, les stratégies d'implantation du patron *Plusieurs vers Plusieurs Personnalisé* généralement établissent des connexions directes entre chaque processus car chaque processus détient un message spécifiquement destiné a un autre processus. Cela n'empêche pas l'utilisation de communications indirectes comme suggéré par Kale [KAL03] et Goldman [GOL05], notamment dans les cas où les messages sont suffisamment petits pour que leur agrégation réduise l'impact du coût d'initialisation des messages.

Toutefois, l'un des grands facteurs qui influencent la performance des opérations de type *All-to-All* est la congestion du réseau. À cause de l'important flux d'échange de données simultané entre les processus, le réseau d'interconnexion est fortement sollicité, ce qui peut augmenter considérablement le temps d'exécution de ces opérations, spécialement si les liens et commutateurs atteignent leurs limites d'utilisation.

À cause des effets néfastes de la congestion du réseau, plusieurs travaux visent le contrôle des communications pour réduire les effets de la congestion. Des travaux sur l'ordonnancement des communications, notamment dans les réseaux de type Ethernet commutés, sont apparus récemment, preuve que les opérations de type MPI_Alltoall représentent un problème réel pour ce type de réseau très populaire. Parmi ces travaux, nous pouvons citer ceux de Tam [TAM03] et Faraj [FAR05]. Dans l'approche de Tam, des informations préalablement obtenues sur les performances du réseau et des commutateurs sont utilisées pour calculer la capacité maximale du réseau, et par conséquent, délimiter le débit maximal que chaque processus aura le droit d'utiliser. Comme cette approche requiert une connaissance très précise des performances du réseau, son utilisation reste assez complexe.

Une autre alternative, au contrôle du flux des messages est l'ordonnancement des communications de façon à éviter la surcharge des connexions ; pour cela, le travail de Faraj [FAR05] cherche à éviter que plusieurs messages transitent simultanément entre des sous-réseaux différents. Pour cela, son algorithme identifie le goulot d'étranglement du réseau, et fait l'ordonnancement des messages selon la distinction entre messages « locaux » (à l'intérieur d'un sous-réseau) et « globaux » (celles qui transitent par le goulot d'étranglement). Ainsi, son algorithme cherche à minimiser le nombre de messages globaux simultanés, en même temps que maximiser le nombre de messages transmis à l'intérieur des sous-réseaux.

Cette approche est similaire à celle de Sanders [SAN02] pour les machines multiprocesseurs. Dans le travail de Sanders, le graphe d'interconnexion entre les processus de différents noeuds est factorisé. L'idée est de réduire progressivement le nombre de noeuds qui dépendent des échanges sur le réseau. De cette manière, les échanges entre noeuds différents sont minimisés au fur et à mesure que les étapes de communication se suivent, ce qui réduit aussi le nombre total d'étapes de communication.

5.6.2 MPI_Scan

Un autre exemple d'opération qui suit le patron de communication *Plusieurs vers Plusieurs Personnalisé*, l'opération *MPI_Scan* est aussi utilisé pour faire la réduction des données des processus, à l'exemple de l'opération *MPI_Reduce.* Toutefois, si l'opération *MPI_Reduce* dépose le résultat final seulement dans le buffer du processus *racine*, l'opération *MPI_Scan* permet la distribution des résultats partiels de cette réduction parmi les processus.

En effet, *MPI_Scan* permet que le buffer de réception d'un processus de rang *i* contienne le résultat de la réduction des buffers des processus de rang 0,...,i (inclusif). À part cette différence,

les opérations, leurs sémantiques et leurs contraintes sont similaires à celles de *MPI_Reduce*. Un exemple d'exécution de l'opération *MPI_Scan* est illustré dans la Figure 5.11.

```
MPI_Scan(sendbuf, recvbuf, count, datatype, op, comm)
    IN sendbuf address of send buffer
    OUT recvbuf address of receive buffer
    IN count number of elements in the send buffer
    IN datatype datatype of send buffer elements
    IN op reduce operation
    IN comm communicator
```

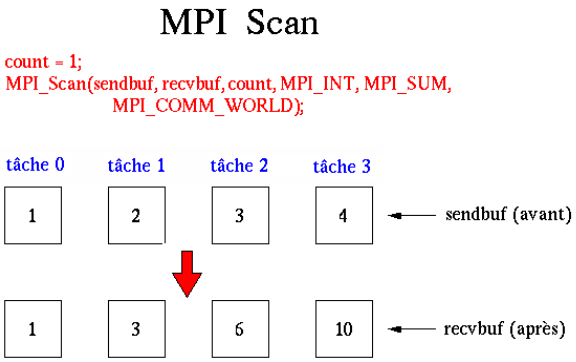


Figure 5.11 *Fonctionnement de l'opération* MPI_Scan *avec une opération de somme*

Étant similaire à l'opération *MPI_Reduce*, *MPI_Scan* peut bénéficier des mêmes optimisations que celle-là. Cependant, comme l'opération *MPI_Scan* oblige que l'ordre des rangs soit respecté, les implantations de *MPI_Scan* doivent préalablement réarranger les processus pour les adapter aux algorithmes optimisés employés [KIE99a].

5.7 Communication avec des messages de taille variable

Jusqu'à maintenant, nous avons présenté des opérations de communication collective qui envoient des messages de même taille. Malgré le fait que les buffers d'envoi et de réception peuvent avoir des tailles différentes, comme dans le cas des *MPI_Gather* et *MPI_Scatter*, les unités de données envoyées ont toujours la même taille.

Certaines applications, comme les fonctions qui utilisent les systèmes linéaires creux, ont néanmoins des données avec des tailles différentes, dont la transmission par les opérations collectives présentées précédemment ne sont pas optimisées. Pour répondre à cette nécessité, la bibliothèque MPI fournit quatre opérations de communication collective qui permettent l'envoi de messages de taille différent, *MPI_Reduce_scatter*, *MPI_Gatherv*, *MPI_Scatterv* et *MPI_Alltoallv*.

5.7.1 MPI_Reduce_scatter

Une variante du patron *Plusieurs vers Plusieurs Personnalisé*, l'opération *MPI_Reduce_scatter* permet la redistribution des résultats individualisés après la réduction. La différence par rapport à l'opération *MPI_Allreduce* est que celle-ci diffuse le résultat de la réduction entre tous les processus, alors que l'opération *MPI_Reduce_scatter* fait la répartition des données selon le rang des processus. De plus, l'opération *MPI_Reduce_scatter* permet que chaque processus reçoive des messages

de tailles différentes, indiqués dans le tableau *recvcounts*. Son fonctionnement est illustré par Figure 5.12.

```
MPI_Reduce_scatter(sendbuf, recvbuf, recvcounts, datatype, op, comm)
    IN sendbuf address of send buffer
    OUT recvbuf address of receive buffer
    IN recvcounts integer array
    IN datatype datatype of send buffer elements
    IN op reduce operation
    IN comm communicator
```

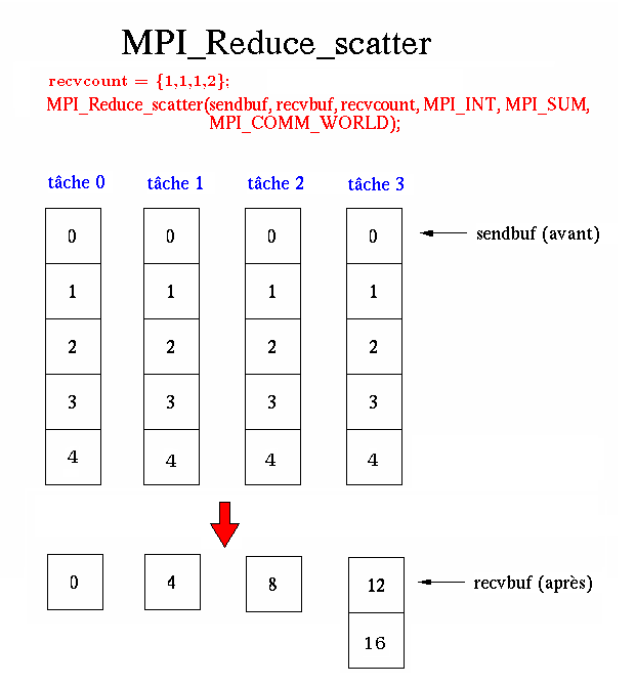


Figure 5.12 *Fonctionnement de l'opération* MPI_Reduce_scatter

Pour implanter l'opération *MPI_Reduce_scatter*, la plus grande difficulté est le traitement des messages de taille différente. En effet, selon la définition du patron MPI, son fonctionnement est équivalent à un *MPI_Reduce* dont le paramètre *count* est la somme des valeurs de *recvcounts[i]*, suivi d'un *MPI_Scatterv*. Cette approche est suivie par Chan [CHA04], LAM-MPI [LAM04] et les anciennes versions de MPICH [THA03, THA05]. Cependant, la performance de cette approche reste simplement la somme des temps d'exécution des différentes parties, ce qui n'est pas toujours optimal.

Des efforts pour établir des implantations spécifiques pour le *Reduce-scatter* ont été faits pour Iannello *et al.* [IAN97, BER03]. Dans leurs travaux, deux algorithmes distincts sont proposés, l'un pour les opérations commutatives et l'autre pour les opérations non-commutatives. Toutefois, les algorithmes de Iannello dépendent d'un paramètre λ, obtenue à partir du calcul de l'index d'une fonction Fibonacci généralisée.

Tan [TAN02] et Thakur [THA03, THA05], par contre, ont concentré leurs efforts sur l'utilisation de fonctions efficaces d'échange de données entre les processus. Si le travail de Tan est celui qui

présente le plus d'alternatives d'implantation, il n'est pas complet car ils ont utilisé des *recvcounts* de même taille.

Dans le travail de Thakur *et al.* [THA03, THA05], différents algorithmes sont utilisés selon la taille des messages et la nature des opérations de réduction. Ainsi, pour des petits messages, un algorithme de type *recursive-halving* (similaire au *recursive-doubling* utilisé par l'*Allgather*) est utilisé si les opérations sont commutatives. La structure de communication de cet algorithme, présenté dans la Figure 5.13, est plus efficace quand le nombre de processus est une puissance de 2, similairement aux observations faites pour le *Allgather*.

Malheureusement, cet algorithme ne peut être utilisé que si les opérations de réduction sont commutatives, à moins que les données puissent être préalablement réarrangées. Dans le cas des opérations non-commutatives, Thakur *et al.* ont adopté un algorithme similaire à *Allgather*, où les processus collectent tous les messages (même ceux qui ne leur sont pas adressés), et effectuent la réduction localement.

Bien sûr, ce dernier algorithme n'est efficace que si les messages sont réellement petits ; par conséquent, leur implantation bascule vers un échange bidirectionnel dés que la taille des messages dépasse les 512 octets, alors que l'algorithme *recursive-halving* (utilisé pour les opérations commutatives) reste efficace jusqu'à 512 Koctets environ, lorsque lui aussi est remplacé par l'algorithme d'échange bidirectionnel.

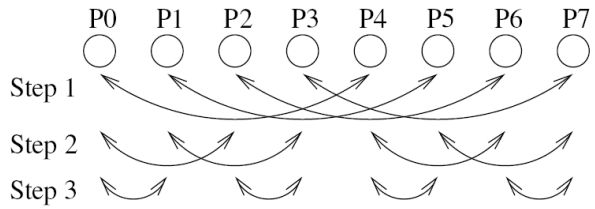


Figure 5.13 *Structure de communication pour un algorithme* recursive-halving

5.7.2 MPI_Gatherv

MPI_Gatherv est une opération de communication collective de type *Plusieurs vers Un Personnalisé* comme l'opération *MPI_Gather*, mais qui permet l'envoi de messages avec des tailles variables. Pour cela, *MPI_Gatherv* remplace le paramètre *count* pour un tableau *recvcounts,* qui décrit la taille des messages envoyés par chaque processus. Par ailleurs, la flexibilité de cette opération est augmentée grâce au paramètre *displs*, qui décrit le placement des données dans le buffer du processus *racine*, comme illustré par la Figure 5.14 (avec *MPI_Gather*, les données étaient enregistrées selon le rang des processus). La définition des paramètres de MPI_Gatherv est la suivante :

```
   MPI_Gatherv(sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype,
root, comm)
       IN sendbuf starting address of send buffer
       IN sendcount number of elements in send buffer
       IN sendtype datatype of send buffer elements
       OUT recvbuf address of receive buffer
       IN recvcounts integer array
       IN displs integer array of displacements
       IN recvtype data type of recv buffer elements
       IN root rank of receiving process
       IN comm communicator
```

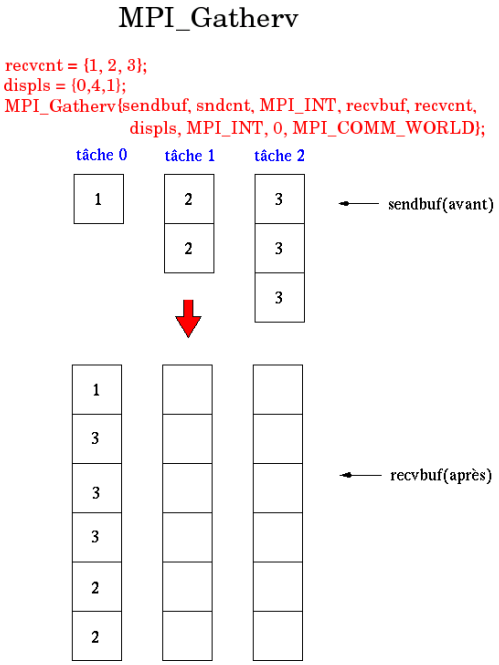


Figure 5.14 *Fonctionnement de l'opération* MPI_Gatherv

Comme résultat, l'opération *MPI_Gatherv* agit comme si chaque processus, y compris le processus racine, envoyait un message *MPI_Send(sendbuf, sendcount, sendtype, root, ...)* - avec des messages de tailles différentes - et le processus racine exécutait *n* fois l'opération *MPI_Recv(recvbuf+displs[i]-extent(recvtype), recvcounts[i], recvtype, i, ...)*.

De cette manière, les données envoyées par le processus *i* sont enregistrées à la position i du buffer du processus racine, position dont l'adresse de début est définie par le paramètre *displs[i]*.

Étant directement liée aux méthodes d'implantation de *MPI_Gather*, l'opération *MPI_Gatherv* peut bénéficier des mêmes techniques d'optimisation. En effet, la littérature ne présente quasiment aucune référence à des techniques d'implantation spécifiques pour le *MPI_Gatherv*, donnant préférence au problème plus généralisé représenté par l'opération *MPI_Alltoallv*.

5.7.3 MPI_Scatterv

MPI_Scatterv est l'opération inverse de *MPI_Gatherv*. Elle permet l'extension des fonctionnalités de *MPI_Scatter* en permettant la distribution de données avec des tailles différentes, décrites dans le tableau *sendcounts*. De plus, le paramètre *displs* permet que les données soient arrangées dans le buffer d'envoi du processus *racine* de manière plus flexible. Un exemple de son fonctionnement est donné par la Figure 5.15.

```
   MPI_Scatterv(sendbuf, sendcounts, displs, sendtype, recvbuf, recvcount, recvtype,
root, comm)
       IN sendbuf address of send buffer
       IN sendcounts integer array
       IN displs integer array of displacements
       IN sendtype datatype of send buffer elements
       OUT recvbuf address of receive buffer
```

```
IN recvcount number of elements in receive buffer
IN recvtype data type of recv buffer elements
IN root rank of sending process
IN comm communicator
```

À notre connaissance, *MPI_Scatterv* ne bénéficie d'aucune technique d'implantation spécifique ; les travaux dans la littérature sont orientés vers le problème de l'opération MPI_Alltoallv, plus général.

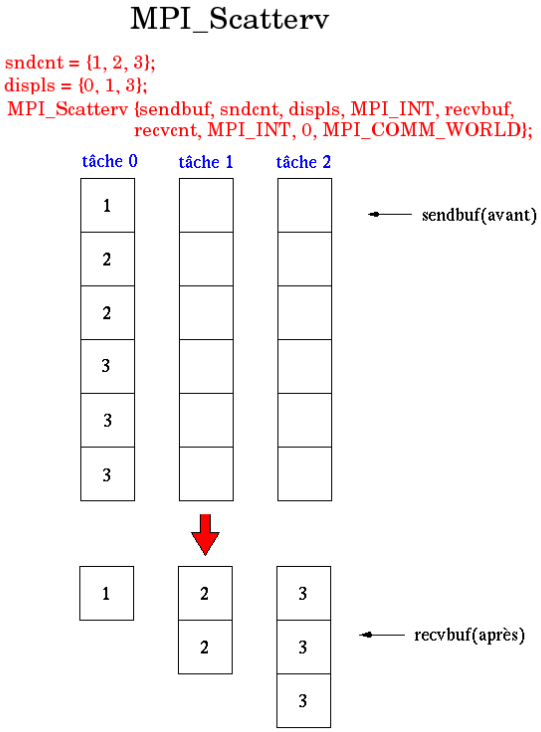


Figure 5.15 *Fonctionnement de l'opération* MPI_Scatterv

5.7.4 MPI_Allgatherv

MPI_Allgatherv est une opération de communication collective de type *Plusieurs vers Plusieurs* comme l'opération *MPI_Allgather*, mais qui permet l'envoi de messages avec des tailles variables (voir Figure 5.16). Pour cela, *MPI_Allgatherv* remplace le paramètre *count* pour un tableau *recvcounts,* qui décrit la taille des messages de chaque processus. Par ailleurs, la flexibilité de cette opération est augmentée grâce au nouveau paramètre *displs*, qui décrit le placement des données dans le buffer des processus (avec MPI_Allgather, les données étaient placées selon le rang des processus). La définition des paramètres de MPI_Allgatherv est la suivante [SNI96] :

```
   MPI_Allgatherv(sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype,
comm)
       IN sendbuf starting address of send buffer
       IN sendcount number of elements in send buffer
```

```
IN sendtype datatype of send buffer elements
OUT recvbuf address of receive buffer
IN recvcounts integer array
IN displs integer array of displacements
IN recvtype data type of recv buffer elements
IN comm communicator
```

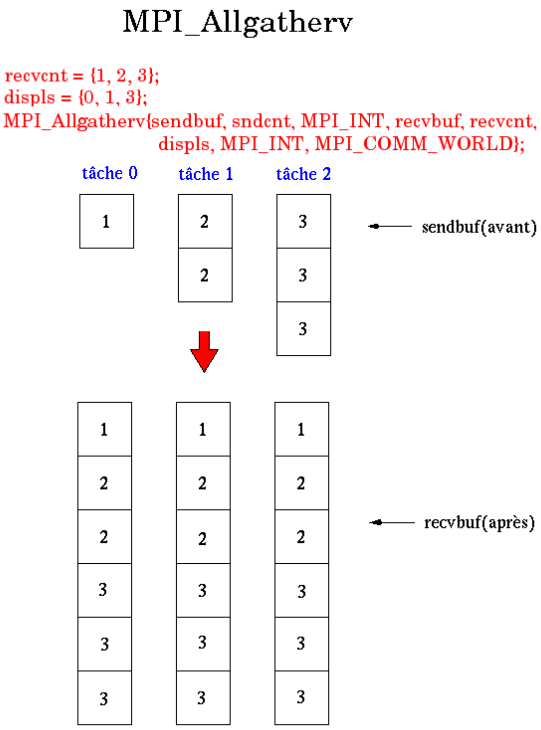


Figure 5.16 *Fonctionnement de l'opération* MPI_Allgatherv

5.7.5 MPI_Alltoallv

MPI_Alltoallv augmente la flexibilité de l'opération *MPI_Alltoall* par la possibilité de définir préalablement non seulement la taille des messages à envoyer, mais aussi la localisation initiale de ces données (par le biais du paramètre *sdispls*) et l'adresse où ces données seront enregistrées (paramètre *rdispls*).

```
   MPI_Alltoallv(sendbuf, sendcounts, sdispls, sendtype, recvbuf, recvcount, rdispls,
recvtype, comm)
       IN sendbuf address of send buffer
       IN sendcounts integer array
       IN sdispls integer array of send displacements
       IN sendtype datatype of send buffer elements
       OUT recvbuf address of receive buffer
       IN recvcounts integer array
       IN rdispls integer array of recv displacements
       IN recvtype data type of recv buffer elements
```

```
IN comm communicator
```

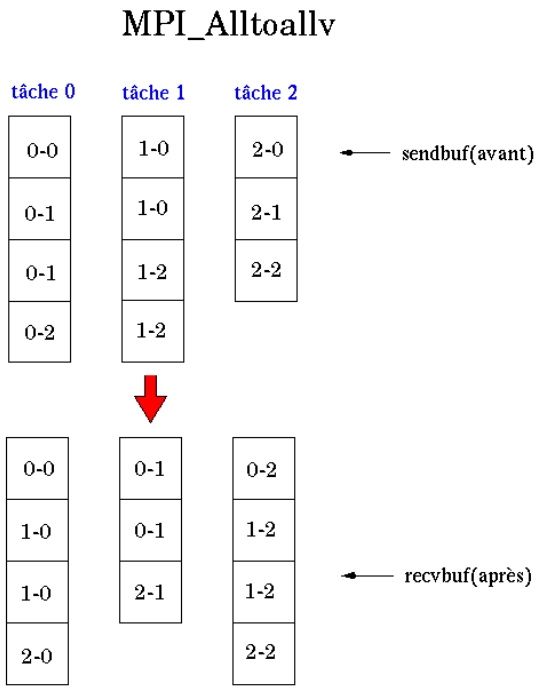


Figure 5.17 *Fonctionnement de l'opération* MPI_Alltoallv

En effet, la définition de *MPI_Alltoallv* donne autant de flexibilité que si les processus effectuaient *n* communications point à point indépendantes, à l'exception que tous les messages ont le même type de données, comme démontre la Figure 5.17.

Pour cela, le tuple {*sendcount[j]*,*sendtype*} dans le processus *i* doit être équivalent à le tuple {*recvcount[i]*,*recvtype*} dans le processus *j*. Cela implique que la quantité de données envoyées soit la même que la quantité attendue à la réception.

Étant une des opérations les plus générales, le *MPI_Alltoallv* représente un défi important pour l'optimisation des communications. En effet, la possibilité d'envoyer des messages de différentes tailles rend cette opération très dynamique, une fois que l'implantation devient dépendante des messages. Parmi les travaux spécifiques pour le *MPI_Alltoallv*, nous trouvons ceux de Liu [LIW96], Bhat [BHA98] et Goldman [GOL98, GOL02, GOL05]. Le travail de Liu a adressé le problème par la réduction du coût des communications, et a comparé plusieurs stratégies dont la réduction du nombre total de messages, la réduction de la variance des tailles de messages, et la décomposition du groupe de processus en plusieurs niveaux de communication.

Les travaux de Bhat [BHA98] et Goldman [GOL98, GOL02, GOL05], par contre, ont recherché des heuristiques efficaces pour l'ordonnancement des communications. Parmi les stratégies étudiées par Bhat, nous pouvons citer des heuristiques de correspondance (*matching*) et une heuristique basée sur le problème *open shop*. Dans les heuristiques de correspondance, l'algorithme essaye de regrouper les communications en échanges de messages de même taille. Bhat a comparé le comportement de trois variantes de cette heuristique, le *minimum matching* (préférence aux petits messages d'abord), le *maximum matching* (préférence aux grands messages d'abord), et un algorithme glouton,

dont certains processus peuvent être en repos si l'algorithme ne trouve pas un correspondant dont les messages ont la même taille. Dans l'heuristique *open shop*, les processus recherchent un partenaire dés qu'ils sont disponibles pour un échange ; les partenaires sont triés par leur disponibilité, avec un poids plus important pour ceux qui sont déjà disponibles.

Goldman [GOL98, GOL02, GOL05] a aussi étudié plusieurs heuristiques pour l'ordonnancement des communications. Il a d'abord analysé des algorithmes simples, dont l'*échange direct* [BAR05b], où l'*hypercube*, qui requiert $\log_2 P$ étapes de communication et dont à chaque étape les processus échangent tous les messages qu'ils connaissent à ce moment. Ensuite, il s'est intéressé aux heuristiques de correspondance, dont les stratégies *max-min*, *max-weight* et *uniform*. La stratégie *max-min* recherche une série d'échanges dont le poids de la communication la plus petite est maximisé. La stratégie *max-weight* cherche un ordonnancement où la somme des poids des communications est maximisée à chaque étape. Finalement, la stratégie *uniform* cherche des séquences d'échange dont les poids des communications sont identiques. La différence par rapport aux algorithmes de Bhat est que la stratégie *uniform* de Goldman permet la segmentation des messages si l'heuristique ne trouve pas une séquence de communications avec des messages exactement similaires.

D'ailleurs Goldman [GOL05] a constaté que si l'heuristique *uniform* présente les meilleures performances, les dernières étapes de communication sont dédiées à l'échange de petits messages, ce qui n'est pas très efficace à cause du coût de l'heuristique. Goldman a suggéré une approche mixte, dont l'heuristique *uniform* est utilisée pendant un nombre défini d'étapes de communication, et est remplacée pour un algorithme simple comme l'*échange direct* ou l'*hypercube* pour les étapes finales.

5.8 Discussion

Dans ce chapitre, nous avons présenté une synthèse des divers patrons de communication collective existants. Pour mieux illustrer ce sujet, nous avons aussi introduit les fonctions MPI qui reflètent ces patrons de communication. Le choix de la bibliothèque MPI a été intentionnel, car ce standard compte parmi l'un des plus utilisés par les applications parallèles avec mémoire distribuée. De plus, une grande partie des travaux sur l'optimisation des communications collectives utilisent MPI comme référence et environnement de test.

Dans le prochain chapitre nous allons approfondir l'étude sur les opérations collectives, à travers l'analyse et modélisation de trois opérations qui représentent respectivement les patrons de communication *Un vers Plusieurs*, *Un vers Plusieurs Personnalisé* et *Plusieurs vers Plusieurs Personnalisé*. Ces patrons de communication ont été choisis à cause de leur similarité avec d'autres patrons de communication : en général, les techniques utilisées avec ces patrons de communication peuvent être appliquées sur d'autres patrons similaires.

6

Modélisation des Communications Collectives

La modélisation des performances des opérations collectives est une approche très intéressante dans le contexte de notre travail. En effet, nous pouvons utiliser les prédictions pour mieux établir le coût de communication de chaque grappe, et ainsi ordonnancer les communications de manière à minimiser le temps total de diffusion dans la grille de calcul. D'autres travaux s'intéressent à l'ordonnancement de tâches dans un environnement de calcul parallèle ; la connaissance des performances de communication est une information très importante pour la détermination du temps d'exécution d'une application, comme illustré par Midorikawa *et al.* [MID04].

Ce chapitre présente notre expérience dans la construction de modèles de performance qui caractérisent ces patrons de communication collective. Pour illustrer notre approche, ce travail présente des expériences avec les opérations *Broadcast*, *Scatter* et *All-to-All*, lesquelles représentent respectivement les patrons de communications collectives *Un vers Plusieurs*, *Un vers Plusieurs Personnalisé* et *Plusieurs vers Plusieurs Personnalisé*. Conceptuellement simple, les patrons *Un vers Plusieurs* et *Un vers Plusieurs Personnalisé* sont aussi présents sur d'autres opérations comme *Barriers*, *Reduces* et *Gathers*. En revanche, le patron *Plusieurs vers Plusieurs Personnalisé* est beaucoup plus complexe, car il est fréquemment sujet à des troubles dus à la congestion du réseau causés par son flux d'échange de messages.

Ces modèles sont utilisés à la fois pour prédire la performance des opérations et pour choisir la technique d'implantation qui est la mieux adaptée à chaque ensemble de paramètres (nombre de processus, taille des messages, performances du réseau). C'est ainsi que nous avons réalisé des expérimentations sur des environnements réseau Fast Ethernet, Giga Ethernet [SEI98] et Myrinet [BOD95], afin de valider l'efficacité des modèles sous différentes architectures réseau.

6.1 Modèles et définitions

6.1.1 Paramètres de performance

Pour la modélisation des performances des opérations de communication collective, nous avons choisi le modèle de coût *parameterised LogP (pLogP)* [KIE01]. Conformément à ce qui est présenté

dans le chapitre 4, le modèle *pLogP* est une extension du modèle *LogP* qui peut traiter avec précision les petits comme les grands messages, avec une complexité minimale. À cause de cette simplicité, ce modèle permet un prototypage rapide des opérations de communication collective, et les modèles développés avec *pLogP* ont permis la prédiction des performances des communications avec une précision suffisante dans la plupart des cas présentés.

Par conséquent, la terminologie employée dans ce travail utilise ***g(m)*** pour représenter le coût d'envoi d'un message de taille m (le *gap*), ***os(m)*** et ***or(m)*** pour représenter respectivement le surcoût dû à l'envoi et à la réception d'un message de taille m, $\boldsymbol{L}$ pour représenter la latence entre deux noeuds, et $\boldsymbol{P}$ pour représenter le nombre de noeuds. Dans les cas où il y a segmentation des messages, le segment de taille s d'un message m est un multiple de la taille du type basique de données qui est transmis, divisant alors le message initial m en k segments. De même, *g(s)* représente le *gap* d'un segment de taille s.

Pour les expérimentations, nous avons utilisé la bibliothèque LAM-MPI 7.1.2beta [LAM04] ; les expériences consistent en 100 mesures pour chaque ensemble de paramètres (taille du message, numéro de processus), dont la valeur moyenne est considérée dans notre analyse.

Enfin, les expérimentations pratiques de ce chapitre ont été faites sur les grappes **icluster-2** et **Grid eXplorer** (***GdX***). La grappe *icluster-2* est localisée au centre de calcul de l'INRIA Rhône-Alpes[1]. Cette grappe contient 104 ordinateurs Itanium-2 (IA-64, biprocesseur, 900MHz, 3Go) interconnectés pour des réseaux Fast Ethernet et Giga Ethernet [SEI98] commutés et Myrinet [BOD95]. Le système d'exploitation est Red Hat Linux Advanced Server 3.0 avec le noyau version 2.4.21smp. Dans le cadre de cette expérience nous avons utilisé seulement les réseaux Fast Ethernet et Myrinet, car le réseau Giga Ethernet venait d'être installé et était encore très instable. C'est ainsi que nous pouvons observer dans la Figure 6.1 les paramètres pLogP qui caractérisent les réseaux Fast Ethernet et Myrinet de *icluster-2*.

Une première observation est la valeur de la latence. Alors que cette valeur dépend beaucoup de la distance entre deux noeuds, la latence est aussi influencée par l'architecture de communication utilisée, raison pour laquelle le réseau Fast Ethernet présent une latence d'environ 61 microsecondes, et que le réseau Myrinet a une latence de seulement 11 microsecondes.

Nous observons aussi que les valeurs pour g, o_s et o_r ne sont pas toujours linéaires par rapport à la taille des messages : alors que la valeur du gap dans le réseau Fast Ethernet pour un message de 128 Ko est de 0,0114382 s, le gap pour un message de 1Mo est de 0,0894343 s, seulement 7,81 fois plus grand. Cette différences est surtout due à la performance du protocole TCP, qui augmente progressivement le débit et qui par conséquent, est moins performant pour des petits messages que pour des grands messages.

La différence entre les deux architectures réseau est aussi mise en évidence par la différence du gap, où le réseau Fast Ethernet requiert plus de 80 ms pour envoyer un message de 1Mo tandis que le réseau Myrinet nécessite seulement 4 ms.

Il est également intéressant d'analyser l'écart entre g, o_s et o_r. Avec le réseau Fast Ethernet, g et o_s sont pratiquement identiques. Cette première constatation indique que l'opération d'envoi MPI_Send est bloqué jusqu'à la transmission complète du message, ce qui dans le cas de Fast Ethernet inclut le sondage du réseau pour éviter une collision. Ceci explique aussi le fait que o_r soit plus petit, car il dépend seulement de l'accès à la mémoire tampon de l'interface réseau.

Dans le cas du réseau Myrinet l'opération MPI_Send reste aussi bloquée jusqu'à la transmission complète du message. Toutefois, la communication est initialisée directement, ce qui se traduit par une équivalence du gap et des surcoûts d'envoi et réception.

L'autre grappe utilisée dans nos expériences, *Grid eXplorer*, est localisée au centre de calcul de l'IDRIS[2]. Elle contient 216 ordinateurs AMD *opteron* (x86, biprocesseur, 2GHz, 2Go) interconnectés par un réseau Giga Ethernet. Le système d'exploitation est une Debian Sid avec le noyau

[1] http ://i-cluster2.inrialpes.fr/
[2] http ://www.lri.fr/~fci/GdX/

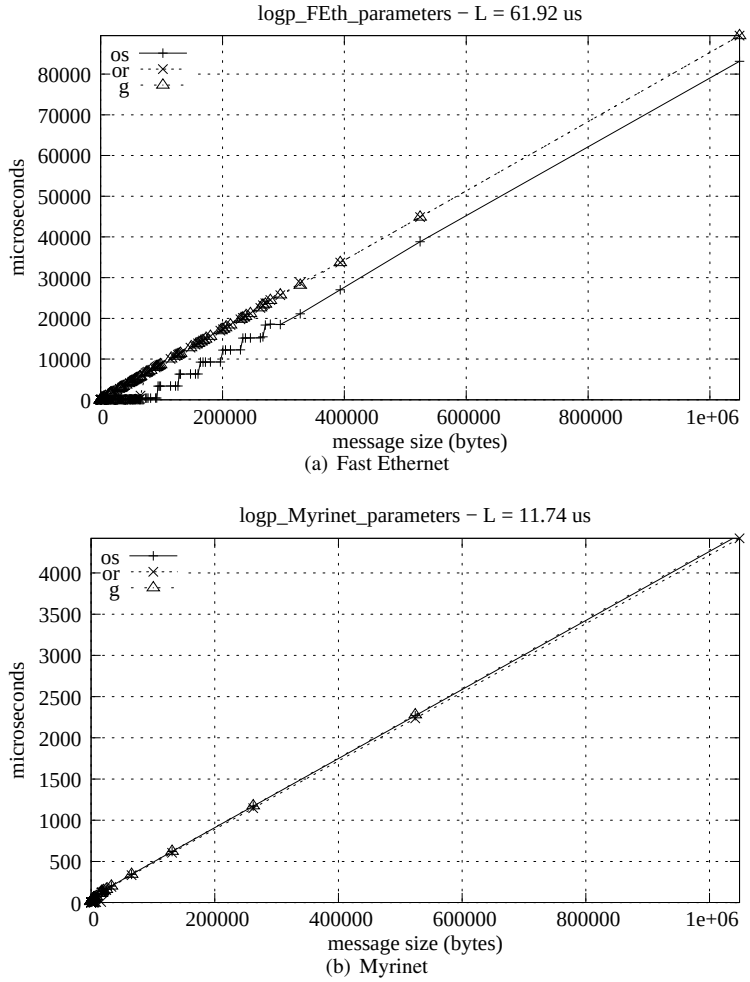


Figure 6.1 *Paramètres* pLogP *des réseaux de la grappe* icluster-2

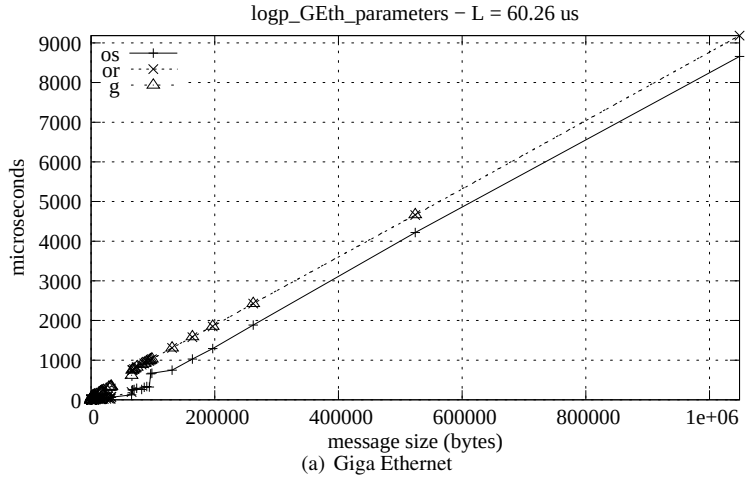
version 2.6.11-smp. Les paramètres pLogP qui caractérisent le réseau Giga Ethernet de la grappe *Grid eXplorer* sont présentés dans la Figure 6.2.

Dans le cas du réseau Giga Ethernet de la grappe *GdX*, le débit est similaire à celui du réseau Myrinet de *icluster-2*, alors que la latence moyenne entre deux noeuds est d'environ 60 microsecondes. Nous observons la différence entre g, o_s et o_r déjà enregistrée sur Fast Ethernet, mais également l'impact du changement de politique de transmission pour des messages plus grands que 64 Ko (Figure 6.3).

6.1.2 Méthodologie de mesure

Pour la mesure des opérations de communication collective, nous avons utilisé une approche similaire à l'approche *Broadcast Barrier*, décrite par Supinski et Karonis [SUP99]. Dans cette approche, la mesure des temps de communication est intercalée avec des opérations de type MPI_Barrier, qui empêchent l'enchaînement des différentes itérations de l'opération de communication collective mesurée, comme illustre l'Algorithme 5.

Cependant, le travail de Supinski indique que cette technique n'est pas suffisamment précise car



(a) Giga Ethernet

Figure 6.2 *Paramètres* pLogP *pour le réseau de la grappe* Grid eXplorer

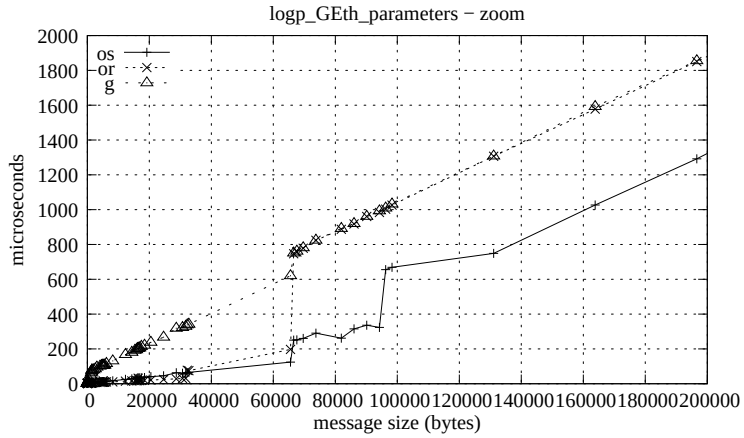


Figure 6.3 *Aperçu des paramètres* pLogP *pour le réseau Giga Ethernet avec des petits message*

Algorithme 5 Algorithme *Broadcast Barrier* de Supinski *et al.* [SUP99]

```
Root (task 0) :
    t1 =current wallclock
    For x = 1 to some large M
        MPI_Bcast
        MPI_Barrier
    t2 =current walltime
    Report (t2 − t1)/M
All other processes :
    For x = 1 to some large M
        MPI_Bcast
        MPI_Barrier
```

le temps de l'opération Barrier est aussi pris en compte dans la mesure du temps, et qui ce temps est difficile de supprimer car l'opération MPI_Barrier est enchaînée à l'opération mesurée.

C'est justement dans la prise en compte du temps du Barrier que notre technique diffère de celle présentée par Supinski. Dans notre cas, nous avons analysé la manière dont l'opération de Barrier est

implanté en MPI, de façon à mieux comprendre le recouvrement des opérations de communication collective.

De manière générale, l'opération de Barrier est implantée comme un double arbre, où initialement les processus indiquent à un processus racine qu'ils sont déjà prêts, et qui ensuite diffuse un signal pour indiquer que tous les processus sont synchronisés. Pour des raisons de performance, ces deux mouvements de données sont implantés comme des arbres binomiaux, à l'exemple des opérations MPI_Reduce et MPI_Bcast. Comme le message diffusé est d'une taille suffisamment réduite (généralement, ce n'est qu'un enveloppe MPI vide), on peut supposer que les temps pour remonter les informations de synchronisation vers le processus racine et la diffusion qui suit sont similaires.

Pour mesurer une opération de communication collective, il est habituel d'utiliser des appels à MPI_Barrier de manière à empêcher l'enchaînement des opérations, ce qui peut altérer le résultat final [SUP99]. Cependant, la barrière a un coût qui est capturé par la mesure, et qui peut influencer dans la précision des résultats, comme indiqué dans la Figure 6.4.

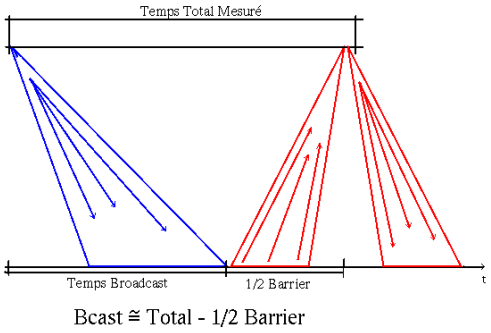


Figure 6.4 *Mesure des temps d'exécution des opérations collectives*

La méthode adoptée dans nos expérimentations considère que ce temps peut être soustrait du temps total mesuré. Pour cela, nous considérons séparément chaque flux de données de la barrière, et son interaction avec les opérations de communication collective. Nous avons pu constater que le temps mesuré par un processus *racine* correspond en réalité à l'opération mesurée plus le premier flux de données de la barrière. À partir de ce constat, notre hypothèse est que le surcoût dû à l'opération MPI_Barrier dans la mesure du temps ne correspond en réalité qu'au temps de la première partie de l'opération MPI_Barrier, alors que la deuxième partie n'est pas comptée dans le temps mesuré.

Nous avons donc imaginé une méthode où le temps de l'opération MPI_Barrier est mesuré préalablement, et dont la moitié de cette valeur est déduite du temps mesuré des opérations collectives, comme indiqué en Algorithme 6.

Ensuite la propre mesure effectuée est utilisée pour valider cette approche : pour cela, le temps mesuré pour l'opération MPI_Bcast avec une petite taille de message doit être proche à la moitié du temps de l'opération MPI_Barrier. Cela peut être observé dans la Figure 6.5, qui présente un exemple de la sortie des mesures de l'opération MPI_Bcast. Comme indiqué, le temps de l'opération MPI_Barrier est suffisamment rapproché du temps du Broadcast pour valider notre approche.

Bien sûr, si la méthode proposée dans ce travail permet une augmentation de la précision des mesures, il existent certaines conditions pour que la mesure soit faite de façon optimale. En effet, notre méthode de mesure est surtout adaptée aux expériences où le processus racine est aussi la racine de la barrière. Comme dans nos expériences nous utilisons l'implantation standard de MPI_Barrier, cela implique que le processus racine des opérations comme le Broadcast et le Scatter sera celui avec le rang 0.

Dans le cas où la racine n'est pas le processus 0, le décalage entre le Barrier et l'opération peut introduire un peu d'imprécision dans la mesure, qui reste toutefois comparable à celle de la stratégie

Algorithme 6 Méthodologie de mesure employée dans ce travail

```
Obtention du temps du barrier :
    MPI_Barrier
    t1 =current wallclock
    MPI_Barrier
    t2 =current walltime
    t_barruer = (t2 − t1)/2

Obtention du temps de l'opération collective :
    MPI_Barrier
    t1 =current wallclock
    MPI_Bcast
    MPI_Barrier
    t2 =current walltime
    T = (t2 − t1 − t_barrier)
```

```
Number of processes = 25
Half Barrier time = 0.000252
Msg Size = 1 bytes
# METHOD,average,stdev,%,max,min
FLAT_TRAD_NOSEG,0.000257,0.000003,1.342433%,0.000264,0.000253
BINARY_TRAD_NOSEG,0.000285,0.000007,2.297346%,0.000303,0.000278
BINOMIAL_TRAD_NOSEG,0.000275,0.000003,1.256119%,0.000282,0.000271
...
```

Figure 6.5 *Exemple des mesures de l'opération MPI_Bcast*

Broadcast-Barrier de Supinski. Une solution possible serait d'implanter l'opération de barrière de manière à ce que le processus racine soit indiqué, mais cela implique le passage d'un paramètre qui ne fait pas partie de la définition de l'interface de MPI_Barrier.

Finalement, pour les opérations qui n'ont pas un seul processus racine (comme par exemple le All-to-All), cette dépendance par rapport à la racine de MPI_Barrier est bien moins importante.

Les prochaines sections détaillent les modèles de communication développés pour les patrons de communication *Un vers Plusieurs*, *Un vers Plusieurs Personnalisé* et *Plusieurs vers Plusieurs Personnalisé*, ainsi que la validation de ces modèles à partir des expérimentations pratiques.

6.2 Un vers Plusieurs *: MPI_Bcast*

Une opération de *Broadcast* s'effectue quand un seul processus, appelé *racine,* envoie le même message de taille m à tous les autres $(P-1)$ processus.

6.2.1 Approches classiques

L'approche classique pour implanter l'opération *Broadcast* utilise des arbres qui sont décrits par deux paramètres, d et h, où d est le nombre maximum de successeurs qu'un noeud peut avoir, et h est la hauteur de cet arbre, le chemin le plus long qui relie la racine et les feuilles de cet arbre. Plus généralement, des arbres de diffusion avec différents degrés d et h peuvent être générés à partir d'un algorithme de type *arbre-alpha* (*alpha-tree* en anglais) suggéré par Bernaschi et Ianello [BER98]. À l'aide de cet algorithme et des paramètres du réseau, un arbre optimal peut être construit à partir des paramètres du réseau et avec d, $h \in [1...P\text{-}1]$ tel que $\sum_{i=o}^{h} d^i \geq P$ soit respecté. Cependant, pour une question de simplicité, la plupart des implantations MPI utilisent des formes fixes telles que les arbres plats ou les arbres binomiaux.

La performance des différentes formes fixes dépend surtout des paramètres du réseau, notamment le *gap*, la latence et le nombre de noeuds. Par conséquent, un réseau avec une latence faible par rapport au *gap* favorise les algorithmes de type Arbre Binaire et Arbre Binomial, qui cherchent à minimiser le temps de communication par la multiplication des sources de transmission. Au contraire, si la latence est trop élevée par rapport au *gap*, les algorithmes de type Arbre Plat sont favorisés, où un seul processus envoie des messages à tous les autres.

De ce fait, la plupart des implantations MPI utilisent deux formes fixes, un Arbre Plat pour un nombre réduit de noeuds (jusqu'à 3 noeuds), et un Arbre Binomial pour un plus grand nombre de noeuds. Cela est dû au fait que généralement les arbres Binomiaux sont optimaux pour les réseaux homogènes locaux (dont une faible latence) ; l'utilisation des arbres Plats ne se fait que pour un nombre très réduit de processus, et cela seulement pour minimiser le coût de construction de l'arbre de diffusion.

À la diversité de formes fixes s'ajoutent aussi différentes techniques d'implantation. En effet, certaines techniques peuvent s'appliquer à des situations spécifiques, comme par exemple, l'envoi de messages avec plus de 64 Ko (valeur par défaut dans la bibliothèque LAM-MPI [LAM04]) ; dans ce cas-là, un message de *rendez-vous* est envoyé préalablement pour préparer le récepteur, réduisant ainsi le stockage des messages dans des buffers temporaires. Autre technique souvent employée est l'utilisation des primitives de communication non bloquantes pour permettre le recouvrement des communications et du calcul. Il faut néanmoins compter que ces techniques apportent un coût supplémentaire aux opérations : alors que le *rendez-vous* ajoute une étape de communication de plus, la communication non bloquante est obtenue à partir de la copie des données vers des buffers intermédiaires. Dans le premier cas le surcoût est constant et correspond à l'envoi de deux messages de taille zéro, tandis que le deuxième cas a un coût qui dépend de la taille du message, et qui correspond à o_s.

À partir des modèles de coût LogP [CUL96a] et pLogP [KIE01] et de travaux comme ceux de Huse [HUS99], Vadhiyar [VAD00] et autres, nous avons déduit les formules qui représentent différentes stratégies de communication évaluées dans ce travail, comme indiqué dans le Tableau 6.1. Certaines de ces stratégies sont clairement inefficaces, comme par exemple le broadcast en Chaîne, qui exécute $P-1$ communications en série. D'autres stratégies, comme les variantes *rendez-vous*, ne sont utilisées qu'à partir d'une taille de message suffisamment grande, ce qui minimise l'impact des étapes de communication supplémentaires dues au *rendez-vous*. Ainsi, nous avons choisi les stratégies d'Arbre Plat et d'Arbre Binomial pour représenter les algorithmes classiques, après avoir vérifié que ceux-ci sont les deux stratégies utilisées par défaut dans l'implantation LAM-MPI [LAM04]. Ces stratégies seront analysées dans la section 6.2.3, où les modèles de communication seront validés à travers des résultats issus des expérimentations pratiques.

Stratégie	**Modèle de Communication**
Arbre Plat	$L + (P-1) \times g(m)$
Arbre Plat Rendez-vous	$3 \times L + (P-1) \times g(m) + 2 \times g(1)$
Chaîne	$(P-1) \times (g(m) + L)$
Chaîne Rendez-vous	$(P-1) \times (g(m) + 2 \times g(1) + 3 \times L)$
Arbre Binaire	$\leq \lceil log_2 P \rceil \times (2 \times g(m) + L)$
Arbre Binomial	$\lceil log_2 P \rceil \times L + \lfloor log_2 P \rfloor \times g(m)$
Arbre Binomial Rendez-vous	$\lceil log_2 P \rceil \times (2 \times g(1) + 3 \times L) + \lfloor log_2 P \rfloor \times g(m)$

TAB. 6.1 *Modèles de communication pour le* Broadcast

6.2.2 Approches par segmentation

Une autre possibilité de construire un *Broadcast* est la composition des chaînes de retransmission [BAT96]. Cette stratégie, possible grâce à la segmentation des messages, présente des avantages importants, comme l'indiquent [KIE01][THA03][BEA04a]. Dans un *Broadcast* Segmentée, la transmission des messages en segments permet le recouvrement de la transmission d'un segment k et la réception du segment k+1, minimisant le *gap*.

Dans ce cas, nous considérons que le segment de taille s d'un message m est un multiple de la taille du type basique de données qui est transmis, divisant alors le message initial m en k segments. Par conséquent, $g(s)$ représente le *gap* d'un segment de taille s. Toutefois, le choix de la taille des segments reste dépendant des caractéristiques du réseau. En effet, l'utilisation de segments trop petits a un surcoût non-négligeable dû à l'en-tête du message, alors que l'utilisation des segments trop grands ne permet pas l'exploitation intégrale du débit du réseau.

La recherche de la taille de segment s qui minimise le temps de communication se fait à l'aide des modèles de communication présentés dans le Tableau 6.2. D'abord, on cherche une taille de segment s qui minimise le temps de communication parmi $s = m/2^i$ pour $i \in [0 \ldots log_2 m]$. Ensuite, on peut affiner la recherche de la taille optimale avec l'aide d'heuristiques comme le « *local hill-climbing* » [KIE01].

Stratégie	Modèle de Communication
Arbre Plat Segmenté	$L + (P-1) \times (g(s) \times k)$
Chaîne Segmentée (Pipeline)	$(P-1) \times (g(s)+L) + (g(s) \times (k-1))$
Arbre Binomial Segmenté	$\lceil log_2 P \rceil \times L + \lfloor log_2 P \rfloor \times g(s) \times k$
Pieuvre avec un degré d	$(d + \lceil \frac{P-(2^d+1)}{(2^d+1)} \rceil) \times (g(s)+L) + (g(s) \times (k-1))$
Scatter/Collection [THA03]	$(log_2 P + P - 1) \times L + 2 \times (\frac{p-1}{p}) \times g(m)$

TAB. 6.2 *Modèles de communication segmentée pour le* Broadcast

Dans les pages suivantes nous décrivons les stratégies qui utilisent la segmentation.

Chaîne segmentée

La stratégie la plus simple de broadcast segmenté est la Chaîne Segmentée, appelée aussi *pipeline*, présentée dans la Figure 6.6. La Chaîne Segmentée présente des performances assez avantageuses quand la taille des messages est considérablement grande, et surtout si la latence est réduite. Malgré ces performances, illustrées dans la Figure 6.21, une implantation en Chaîne Segmentée est sujette à certaines instabilités dues à la variation de performance des différents processus qui composent la chaîne. En effet, il suffit qu'un processus retarde la transmission des segments pour que l'exécution du broadcast soit perturbée.

Pieuvre

Si une chaîne est trop sensible à l'accumulation des retards des processus, une alternative suffisamment efficace est l'implantation d'une stratégie hybride, qui mélange à la fois un Arbre Plat ou Binomial et des chaînes segmentées. Cette stratégie, appelée *pieuvre* (ou *k-chain* en anglais [PJE05], Figure 6.7) utilise un petit « coeur » en forme d'arbre plat ou binomial, dont les processus donnent à leur tour naissance à plusieurs chaînes.

En conséquence, les chaînes formées sont plus courtes et plus nombreuses ; les effets dus au retard d'un processus affectent seulement un groupe de processus, et le retard accumulé sera moins répandu dans les chaînes de diffusion. Ceci dit, un désavantage de cette stratégie par rapport à la

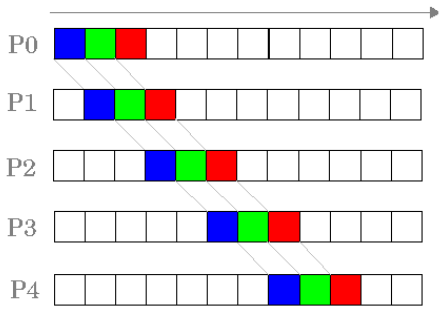


Figure 6.6 *Structure de communication d'une Chaîne Segmentée*

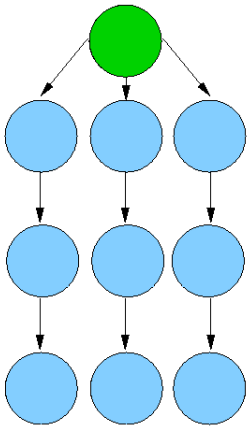


Figure 6.7 *Structure de fonctionnement d'un arbre* pieuvre

Chaîne Segmentée classique est que le nombre de processus doit être suffisamment grand pour que l'accumulation des latences compense le coût initial des arbres de diffusion. Dans les expériences effectuées dans le cadre de ce travail, la performance de la pieuvre n'a pas pu dépasser celle de la chaîne segmentée, malgré un léger gain de performance par rapport à l'arbre binomial. Pour cette raison, et à cause de sa complexité, nous avons préféré l'étude de la chaîne segmentée, technique déjà utilisée pour la distribution de données à large échelle[3].

Scatter/Collection

L'algorithme en arbre binomial est adéquat aux petits messages à cause de son coût de latence logarithmique, tandis qu'un algorithme de type Chaîne Segmentée peut s'avérer efficace pour les grands messages. Cependant, l'algorithme en Chaîne Segmentée requiert une performance uniforme de tous les processus, sinon sa performance est trop sujette à des variations.

Une alternative plus sûre pour la diffusion de grands messages est l'algorithme Scatter-Collection, proposé par Van de Geijn et *al.* [BAT96]. Dans cet algorithme, le message est d'abord partagé entre les processus à l'aide d'une opération similaire à MPI_Scatter ; ensuite, les données sont regroupées

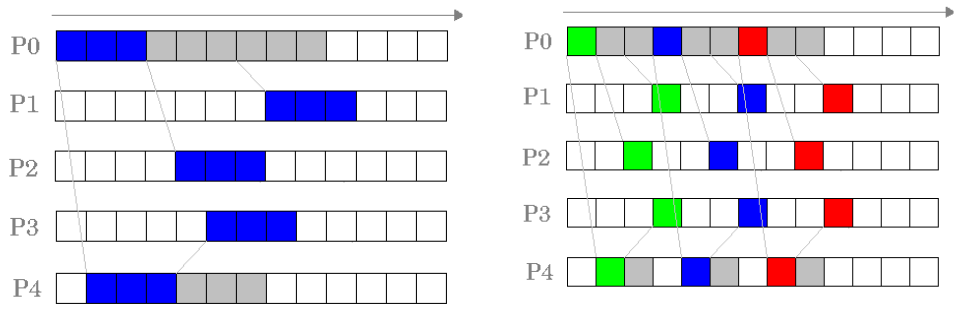
[3] *http ://ka-tools.imag.fr/*

avec une opération similaire à MPI_Allgather. Le temps d'exécution est alors la somme du temps d'exécution du *Scatter* et du *Allgather*.

En effet, Thakur *et al.* [THA03] utilisent cette technique pour l'implantation de MPI_Bcast adaptée aux grands messages, où un algorithme binomial est utilisé pour le *Scatter* et un algorithme en anneau pour le *Allgather*.

Segmentation et arbres de diffusion

Si dans certains cas la segmentation peut augmenter la performance de certaines stratégies de diffusion, cela ne s'applique pas à toutes les stratégies de communication. Par exemple, l'apport de l'utilisation conjointe de la segmentation de messages et des stratégies de broadcast en arbre Plat ou Binomial dépend surtout des paramètres de communication (L et g), une fois que ces stratégies obligent une interdépendance entre processus par rapport à la distribution des données. Pour mieux illustrer cette situation, la Figure 6.8 représentent le schéma de communication pour l'algorithme en arbre binomial avec $L = 1$ et $g = 1$. Nous observons que la structure de cet algorithme oblige un flux de données qui n'est pas adapté à l'enchaînement des communications, comme c'était le cas avec la chaîne segmentée 6.6.



(a) Arbre binomial traditionnel (b) Arbre binomial segmenté

Figure 6.8 *La stratégie en Arbre Binomial sans et avec segmentation avec L=1 et g=1*

En effet, pour des valeurs de $g \geq L$, la chaîne segmentée est la forme optimale ; seulement si $g \leq L$ les autres formes d'arbre deviennent plus intéressantes, car elles rendent possible le recouvrement de la latence de communication par des émissions successives.

6.2.3 Validation des modèles

Pour valider nos modèles de communication, nous avons choisi la comparaison entre les prédictions des modèles et les résultats réels obtenus à partir d'expérimentations sur différentes plates-formes réseaux. Pour illustrer notre approche, nous avons comparé des implantations de MPI_Bcast selon les stratégies Arbre Plat, Arbre Binomial et Chaîne Segmentée. Dans les prochaines pages nous présentons l'analyse des expérimentations effectuées sur chaque plate-forme réseau différente.

Réseau Fast Ethernet

Comme nous pouvons observer dans les Figures 6.9, 6.10 et 6.11, les prédictions des trois stratégies d'implantation représentent presque fidèlement les résultats expérimentaux obtenus sur le réseau Fast Ethernet.

Plus spécifiquement, des différences entre les prédictions et les valeurs réelles sont observées surtout dans le cas de la stratégie en Arbre Binomial, où le temps mesuré pour l'envoi de petits messages est plus élevé que celui prévu, et ne suit pas un comportement linéaire par rapport à la taille des messages. Cette variation de performance, observée aussi dans des expériences avec le réseau Giga Ethernet, a été l'objet d'analyses précédentes (cette discussion peut être retrouvée dans les articles [BAR04a] et [BAR04c]).

D'autres travaux font aussi référence à ces retards, dont un article des développeurs du LAM-MPI [LAM01]. Une enquête profonde, menée par Loncaric [LON00], a indiqué comme l'origine des retards l'implantation des politiques d'acquittement TCP sur Linux, qui retarde des messages même si l'option *socket* TCP_NODELAY est activée.

En effet, Loncaric a constaté que dans certains cas, un seul message à chaque n messages transmis n'est pas acquitté comme il le faut (la valeur de n dépend de la version du noyau Linux utilisé). Cette défaillance du protocole d'acquittement induit un temps supplémentaire nécessaire à la retransmission du message et à son acquittement.

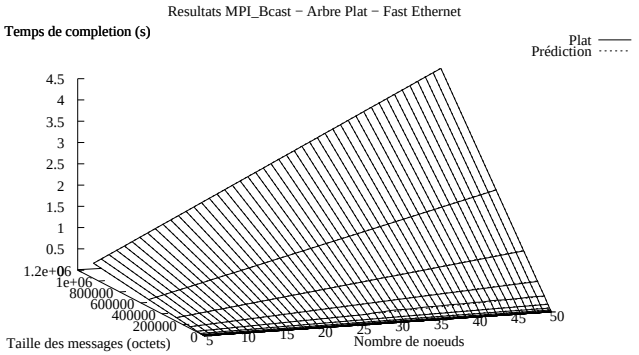


Figure 6.9 *Les performances réelles et prédites pour l'Arbre Plat avec un réseau Fast Ethernet*

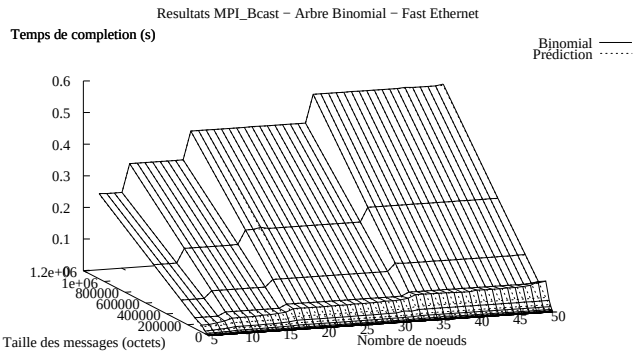


Figure 6.10 *Les performances réelles et prédites pour l'Arbre Binomial avec un réseau Fast Ethernet*

Nous pouvons également observer dans la Figure 6.12 le comportement des trois stratégies et leurs prédictions lorsque varie le nombre de processus. Nous observons que la stratégie en Arbre Plat est beaucoup moins performante que les deux autres stratégies, et sa performance décroît avec l'augmentation du nombre de processus communicants, alors que les autres stratégies sont beaucoup plus stables par rapport au nombre de processus. Cela est du au modèle de communication employé par la stratégie en Arbre Plat, qui la rend directement dépendante du nombre de processus et du surcoût d'envoi des messages (*gap*). Les stratégies en Arbre Binomial et en Chaîne Segmentée sont moins sensibles à l'augmentation du nombre de processus car la première stratégie a un coût loga-

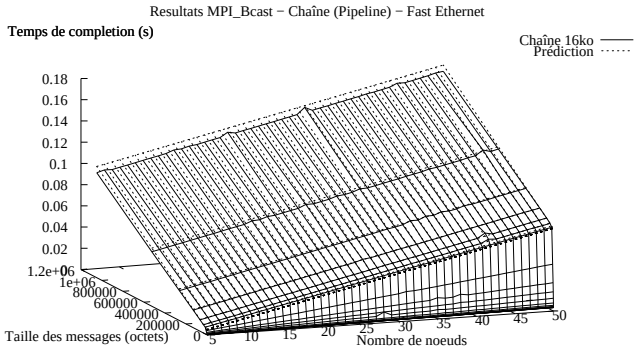


Figure 6.11 *Les performances réelles et prédites pour la Chaîne Segmentée avec un réseau Fast Ethernet*

rithmique, alors que la deuxième est linéairement dépendante de la latence, qui a une valeur bien plus réduite.

Les variations de performance observées dans le cas de la stratégie en Arbre Binomial sont plus visibles dans la Figure 6.13. Ici, nous observons que les résultats réels, normalement très proches des prédictions (à une marge de 10% maximum), s'écartent des prédictions jusqu'à 90% pour des messages autour de 128 Ko. Néanmoins, ces variations affectent des communications où la différence absolue n'est que de quelques millisecondes, ce qui n'empêche pas l'utilisation des modèles de communication pour choisir la meilleure stratégie de communication.

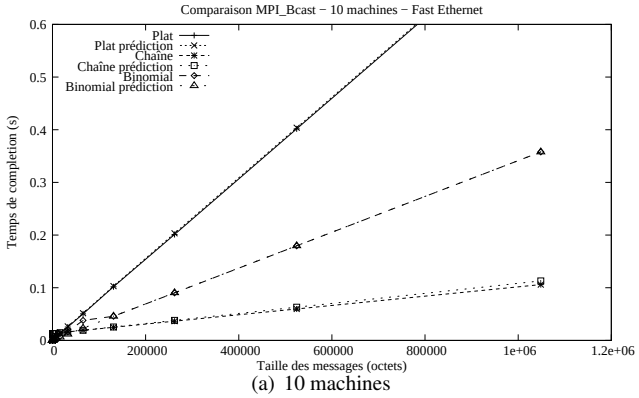
Réseau Giga Ethernet

Les résultats des expérimentations effectuées sur le réseau Giga Ethernet de la grappe *Grid eXplorer* sont présentés dans les Figures 6.14, 6.15 et 6.16. Si les prédictions pour la stratégie en Arbre Plat suivent fidèlement les résultats pratiques, nous observons des variations importantes dans les cas des stratégies en Arbre Binomial et en Chaîne Segmentée. Dans les deux cas, le temps des communications a subi une forte augmentation quand l'application utilise plus de 24 processus. Si l'origine de cette augmentation nécessite une investigation plus approfondie, nous considérons fortement la possibilité d'une surcharge du commutateur. En effet, la stratégie en Arbre Plat limite les communications à la capacité d'envoi du processus *racine*, alors que les autres stratégies favorisent l'envoi simultané de messages par différents processus, ce qui peut surcharger le commutateur.

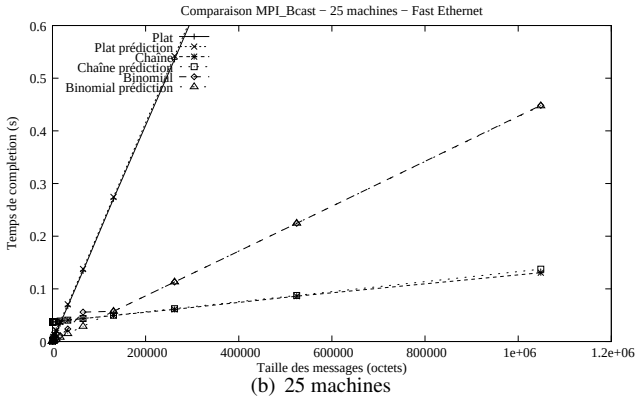
Toutefois, nous pouvons observer dans la Figure 6.17 que si les prédictions ne correspondent pas exactement aux résultats réels, au moins elles suivent le comportement général des trois stratégies d'implantation. Plus exactement, si nous considérons seulement le cas des 10 processus (et que par conséquent sont exemptés des effets dûs au commutateur), les prédictions sont très similaires aux résultats réels, comme atteste la Figure 6.17(a).

Dans tous les cas, nous observons que la stratégie en Arbre Plat est beaucoup moins performante que les deux autres stratégies, et que la Chaîne Segmentée présente la meilleure performance pour les grands messages (la meilleure stratégie pour l'envoi de petits messages dépend surtout du nombre de processus).

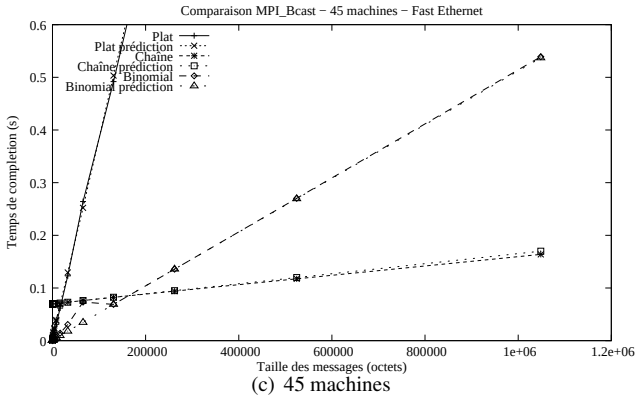
Étant donné que le surcoût dû à la surcharge du commutateur est un facteur difficile à prédire, nous avons concentré l'analyse de l'erreur relative des prédictions au groupe de processus qui n'étaient pas sous l'influence du commutateur surchargé. Ainsi, nous pouvons observer dans la Figure 6.18 que les prédictions sont suffisamment proches des résultats expérimentaux, avec une erreur généralement inférieure à 10%. Nous pouvons aussi constater que l'approche en Arbre Binomial subit les mêmes variations déjà observées dans le cas du réseau Fast Ethernet avec des petits messages, ce qui corrobore l'hypothèse que l'implantation du protocole TCP sous Linux occasionne arbitrairement des retards importants pour certains messages.



(a) 10 machines



(b) 25 machines



(c) 45 machines

Figure 6.12 *Comparaison entre les résultats réels et prédits pour des groupes de 10, 25 et 45 machines dans un réseau Fast Ethernet*

Réseau Myrinet

Les expériences conduites sur le réseau Myrinet de la grappe *icluster-2* sont présentées dans les Figures 6.19, 6.20 et 6.21. Similairement à ce qui a déjà été observé pour les réseaux Fast Ethernet et Giga Ethernet, les modèles de communication des stratégies en Arbre Plat et Arbre Binomial reproduisent fidèlement le comportement des expériences pratiques.

Dans le cas de la Chaîne Segmentée toutefois les prédictions sous-estiment le temps nécessaire à l'exécution du broadcast. Nous croyons que l'erreur observée est liée au coût de la manipulation des segments. Ce coût, qui normalement représente une petite partie du temps total d'exécution, devient plus important quand l'environnement réseau utilisé est suffisamment rapide, à l'exemple du réseau

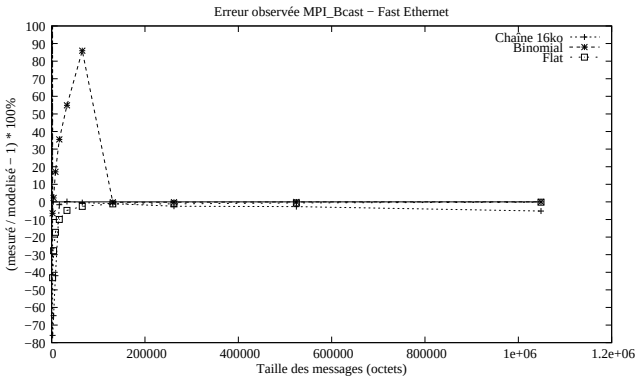


Figure 6.13 *L'erreur des prédictions par rapport aux valeurs mesurées, dans un réseau Fast Ethernet*

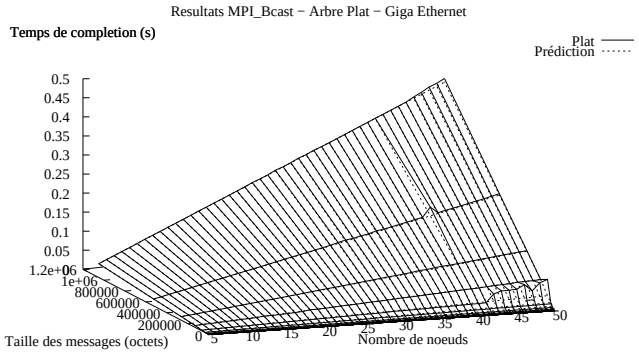


Figure 6.14 *Les performances réelles et prédites pour l'Arbre Plat avec un réseau Giga Ethernet*

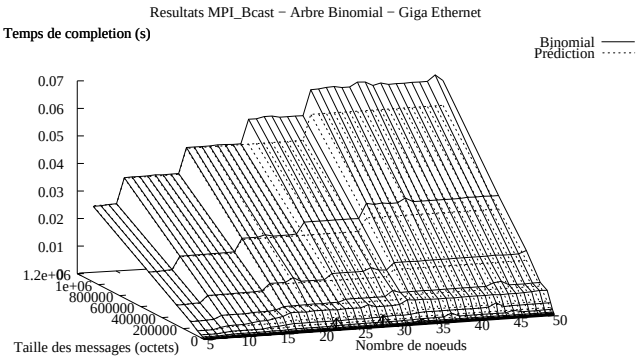


Figure 6.15 *Les performances réelles et prédites pour l'Arbre Binomial avec un réseau Giga Ethernet*

Myrinet, où la latence et le *gap* sont très faibles par rapport aux autres architectures réseau utilisées. Pour vérifier si cette erreur est réellement due à la manipulation des segments et pas au mauvais choix du segment, nous avons effectué l'expérience avec d'autres tailles de segment, 32Ko et 64Ko (Figures 6.22a et 6.22b). Celles ci montrent d'un côté que le surcoût par rapport à la prédiction reste importante, mais aussi que l'utilisation de segments plus grands n'a pas permis la réduction du temps de communication.

Bien que les prédictions pour la Chaîne Segmentée ont une marge d'erreur plus importante, nous pouvons observer dans la Figure 6.23 que les prédictions de cette stratégie sont encore suffisamment

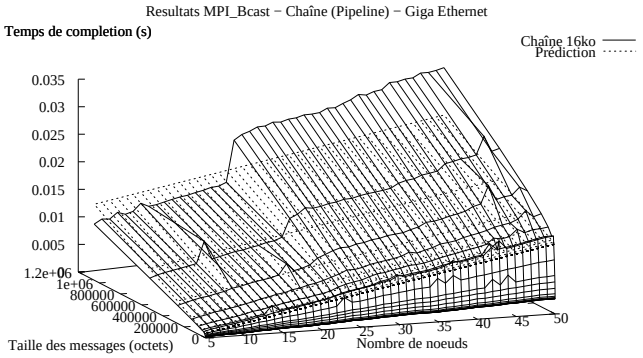


Figure 6.16 *Les performances réelles et prédites pour la Chaîne Segmentée avec un réseau Giga Ethernet*

proches pour permettre le choix de la meilleure stratégie à utiliser. Cependant, avec le développement et la démocratisation d'architectures réseau plus rapides, on peut envisager l'intégration de ce coût aux modèles de communication, à l'exemple de l'approche proposée pour le patron de communication *Plusieurs vers Plusieurs Personnalisé*, décrite dans la section 6.4.

L'impact du coût dû à la segmentation des messages sur l'approche Chaîne Segmentée peut être mieux observé à travers l'analyse de l'erreur présentée dans la Figure 6.24. Alors que les autres stratégies de communication ont une erreur insignifiante, l'erreur des prédictions pour la Chaîne Segmentée peut dépasser les 30% selon la taille de segment utilisée, mais se stabilise avec l'augmentation de la taille des messages envoyés.

6.3 Un vers Plusieurs Personnalisé : *MPI_Scatter*

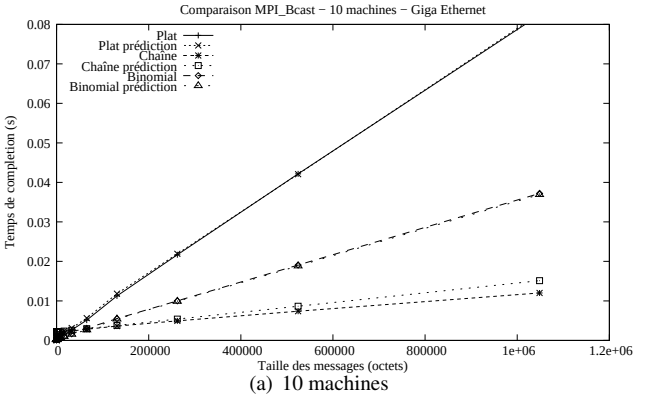
L'opération *Scatter*, aussi appelée « *broadcast* personnalisé », est une opération où le processus *racine* détient P messages différents de taille m qui seront distribués également entre tous les P processus. Parce que le *Scatter* est l'opération symétrique de l'opération *Gather*, les modèles développés pour le *Scatter* peuvent aussi représenter le patron de communication « plusieurs vers un » de l'opération *Gather*.

Dans le cas du *Scatter*, où la *racine* détient un message différent pour chaque processus, il est généralement considéré que le meilleur algorithme pour les réseaux homogènes utilise les Arbres Plats [KIE01]. Par conséquent, l'implantation en Arbre Plat, illustrée dans la Figure 6.25, est l'approche par défaut des bibliothèques MPI.

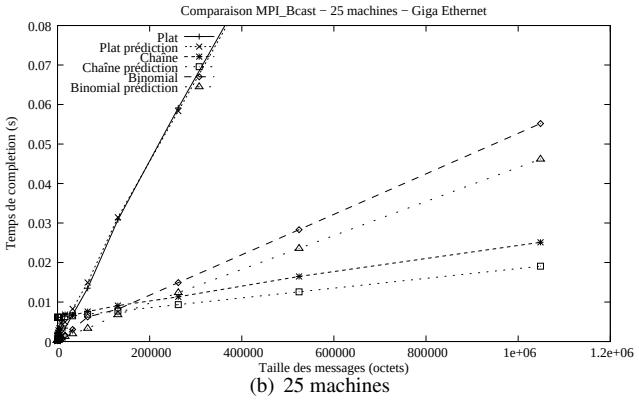
Ce choix est dû au fait que des alternatives pour les Arbres Plats requièrent toujours que des grands ensembles de messages soient transmis par des noeuds intermédiaires. En prenant par exemple le cas des Arbres Binomiaux, le processus *racine* transmet à ses successeurs des paquets de messages qui contiennent plusieurs messages (conformément à la Figure 6.26). Si d'un côté cette stratégie peut bénéficier des envois parallèles, elle a des inconvénients car la transmission des paquets de messages nécessite plus de temps qu'un seul message. Par conséquent, l'efficacité des Arbres Binomiaux dépend surtout de la vitesse de transmission des grands messages, et on observe alors l'effet du compromis entre les envois parallèles et la transmission des grands messages sur le temps total de l'opération.

De la même manière, un Scatter structuré sous la forme d'une chaîne oblige la retransmission de plusieurs messages non-adressés au processus (Figure 6.26). Une fois que chaque message est différent, nous ne pouvons pas utiliser des techniques de segmentation, et le coût non-négligeable de cette approche ne justifie pas son utilisation.

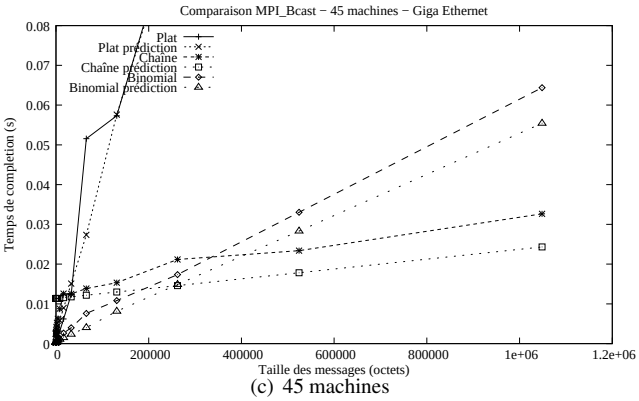
Le Tableau 6.3 présente les modèles de communication développés pour le *Scatter*. Alors que le coût de la Chaîne est bien plus élevé que celui du Scatter en Arbre Plat, la performance de l'al-



(a) 10 machines



(b) 25 machines



(c) 45 machines

Figure 6.17 *Comparaison entre les résultats réels et prédits pour des groupes de 10, 25 et 45 machines dans un réseau Giga Ethernet*

gorithme en Arbre Binomial semble plus proche de celle de l'algorithme en Arbre Plat. En effet, la performance de l'Arbre Binomial dépend de la linéarité des valeurs de g : si le coût d'envoi de grands messages est proportionnellement plus petit que celui des petits messages, le modèle en Arbre Binomial pourrait avoir un léger avantage. Cependant, le coût d'implantation d'un tel algorithme est bien plus élevé, raison pour laquelle nous avons choisi uniquement l'approche par défaut de MPI pour valider nos modèles de communication.

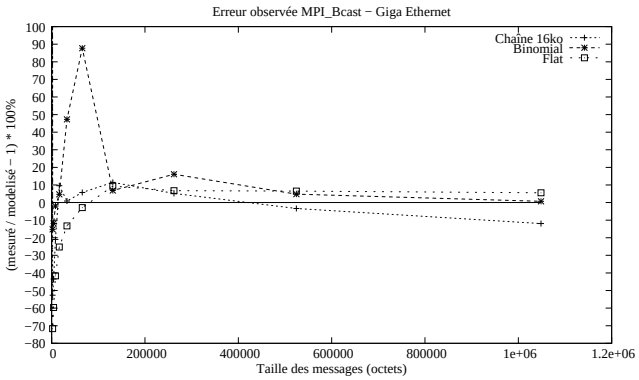


Figure 6.18 *L'erreur des prédictions par rapport aux valeurs mesurées, dans un réseau Giga Ethernet*

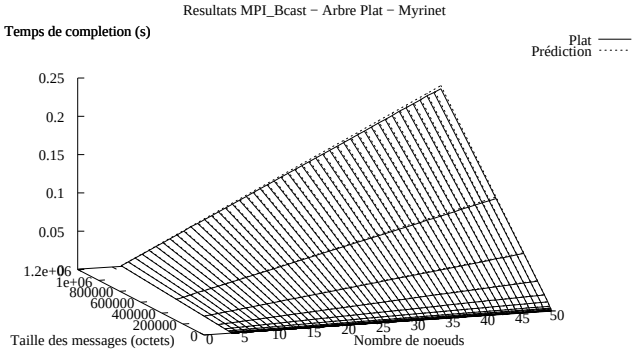


Figure 6.19 *Les performances réelles et prédites pour l'Arbre Plat avec un réseau Myrinet*

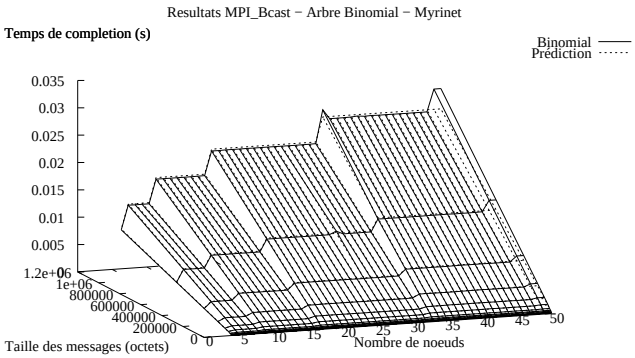


Figure 6.20 *Les performances réelles et prédites pour l'Arbre Binomial avec un réseau Myrinet*

6.3.1 Validation des modèles

Réseau Fast Ethernet

Les expériences conduites sur le réseau Fast Ethernet de la grappe *icluster-2* sont présentées dans la Figure 6.27. Ces images démontrent que les prédictions pour la stratégie en Arbre Plat reproduisent fidèlement le comportement des expériences pratiques. De même, l'erreur des prédictions est très réduite pour les grands messages, généralement à moins de 10%, comme l'atteste la Figure 6.28. Pour les petits messages, la marge d'erreur est un peu plus élevée, mais encore petite si on

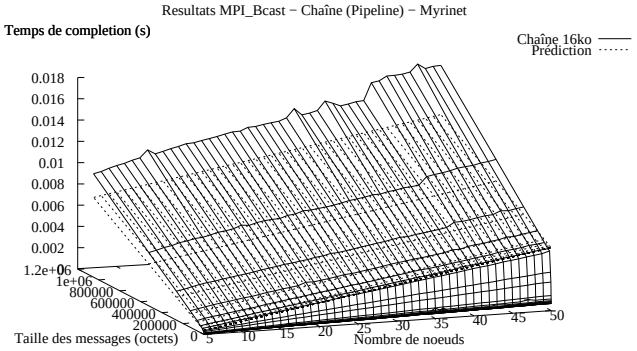
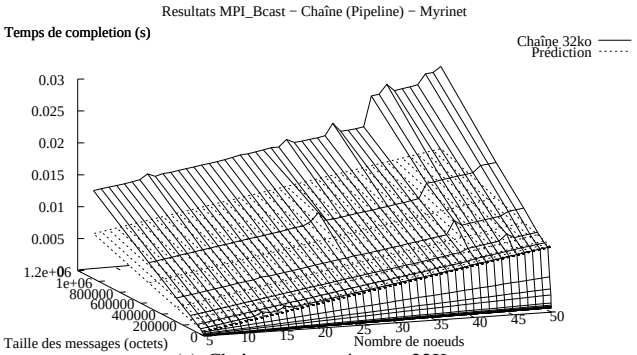
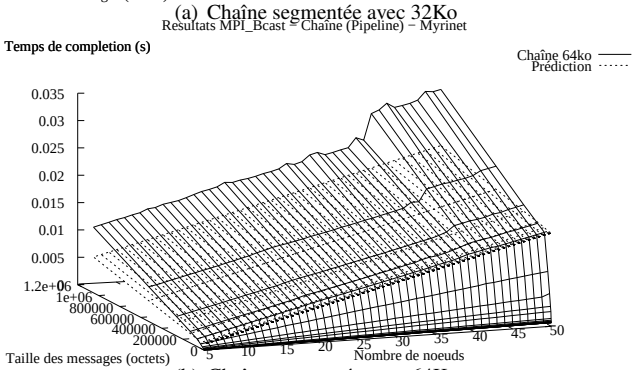


Figure 6.21 *Les performances réelles et prédites pour la Chaîne Segmentée avec un réseau Myrinet*



(a) Chaîne segmentée avec 32Ko

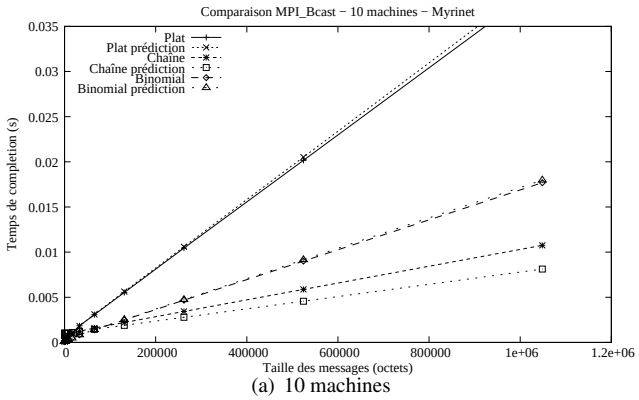


(b) Chaîne segmentée avec 64Ko

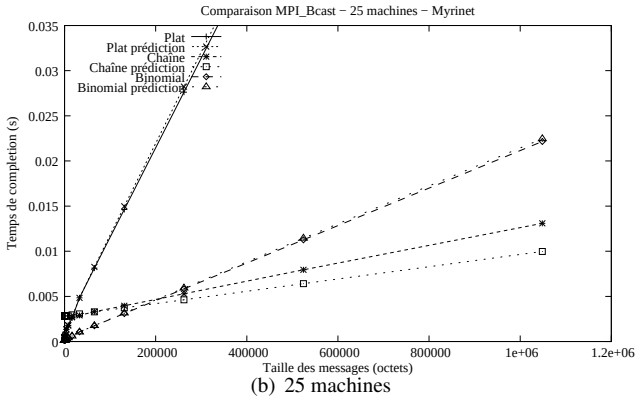
Figure 6.22 *Les performances réelles et prédites pour la Chaîne Segmentée sur un réseau Myrinet avec différentes tailles de segment*

Stratégie	**Modèle de Communication**
Arbre Plat	$L + (P-1) \times g(m)$
Chaîne	$(P-1) \times L + \sum_{j=1}^{P-1} g(j \times m)$
Arbre Binomial	$\leq \lceil log_2 P \rceil \times L + \sum_{j=0}^{\lfloor log_2 P \rfloor - 1} g(2^j \times m)$ $\approx \lceil log_2 P \rceil \times L + (P-1) \times g(m)$

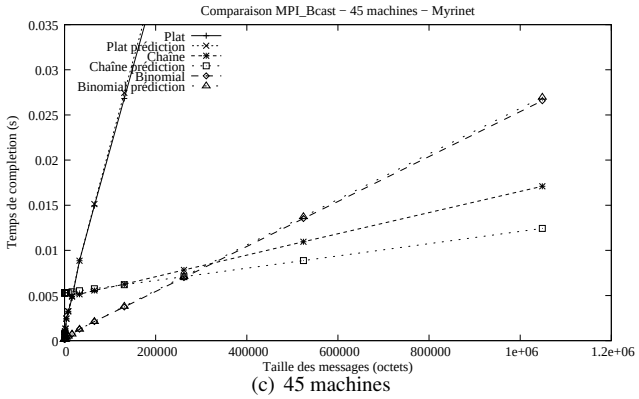
TAB. 6.3 *Modèles de Communication pour le* Scatter



(a) 10 machines



(b) 25 machines



(c) 45 machines

Figure 6.23 *Comparaison entre les résultats réels et prédits pour des groupes de 10, 25 et 45 machines dans un réseau Myrinet*

considère sa valeur absolue.

Réseau Giga Ethernet

Similairement aux expériences sur Fast Ethernet, les valeurs mesurées sur le réseau Giga Ethernet indiquent que les prédictions pour la stratégie en Arbre Plat correspondent aux valeurs réelles mesurées dans cette architecture réseau, comme nous pouvons l'observer dans la Figure 6.29.

D'ailleurs, la performance du réseau se reflète dans le temps de communication. Avec un débit

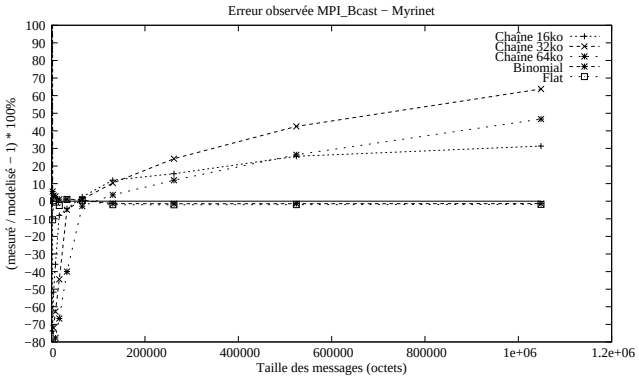


Figure 6.24 *L'erreur des prédictions par rapport aux valeurs mesurées, dans un réseau Myrinet*

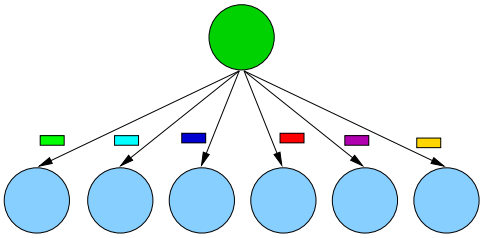


Figure 6.25 *Stratégie en Arbre Plat pour le Scatter*

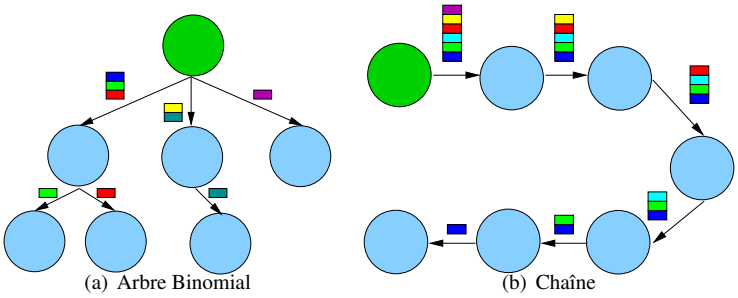


Figure 6.26 *Stratégies en Arbre Binomial et Chaîne pour le Scatter*

plus élevé, le réseau Giga Ethernet obtient une performance presque 10 fois supérieure à celle du réseau Fast Ethernet.

L'analyse de l'erreur des prédictions (Figure 6.30) montre que les valeurs obtenues à partir des expériences sont généralement très proches des prédictions, avec une marge d'erreur très proche de zéro. En effet, si les prédictions semblent moins précises dans le cas des petits messages, il faut considérer aussi que la moindre variation a un impact plus important sur l'erreur, vu la durée de ces communications.

Réseau Myrinet

Encore une fois, le modèle de coût établi pour le Scatter en Arbre Plat reproduit fidèlement le comportement des expériences pratiques, comme démontre la Figure 6.31.

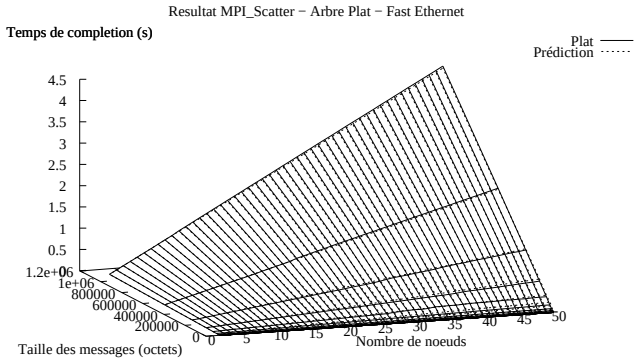


Figure 6.27 *Performances réelles et prédites pour le* Scatter *en Arbre Plat avec un réseau Fast Ethernet*

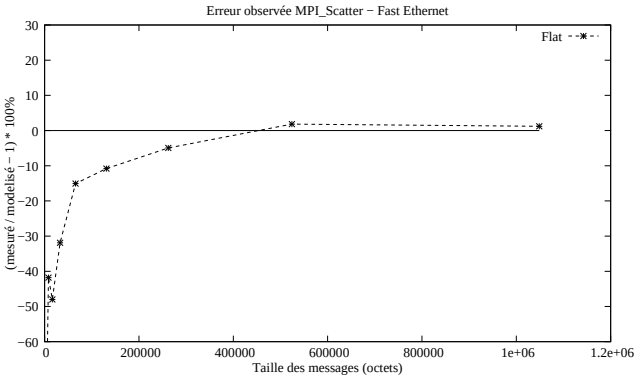


Figure 6.28 *Erreur des prédictions de l'opération MPI_Scatter pour l'envoi de messages de 1Mo*

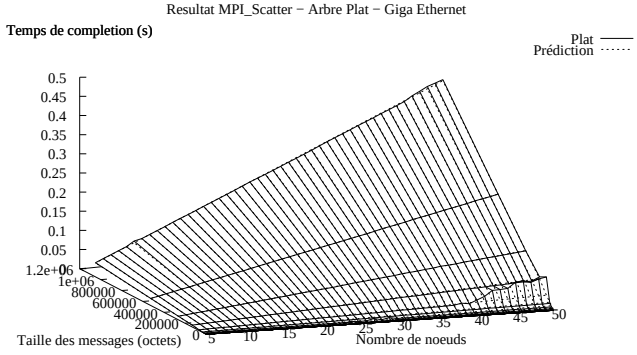


Figure 6.29 *Performances réelles et prédites pour le* Scatter *en Arbre Plat avec un réseau Giga Ethernet*

De plus, comme le réseau Myrinet utilisé est bien plus stable que les réseaux Fast Ethernet et Giga Ethernet notamment, la marge d'erreur entre les prédictions et les expériences est très réduite, et démontre l'adéquation du modèle établi dans le but d'évaluer et prédire la performance de l'opération MPI_Scatter.

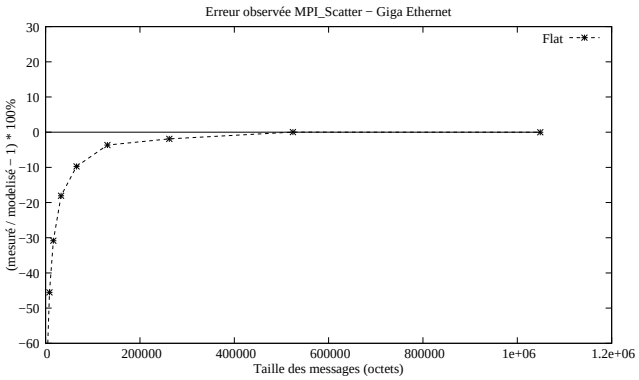


Figure 6.30 *Erreur des prédictions de l'opération MPI_Scatter pour l'envoi de messages de 1Mo*

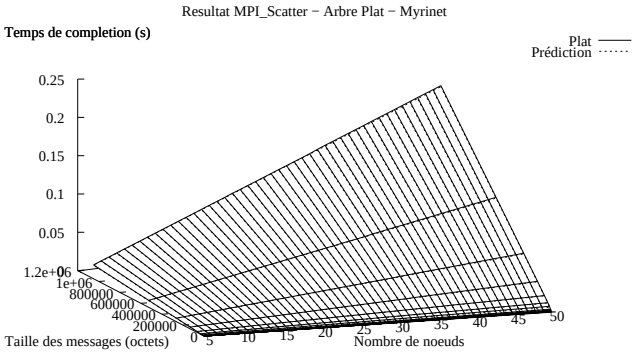


Figure 6.31 *Performances réelles et prédites pour le* Scatter *en Arbre Plat avec un réseau Myrinet*

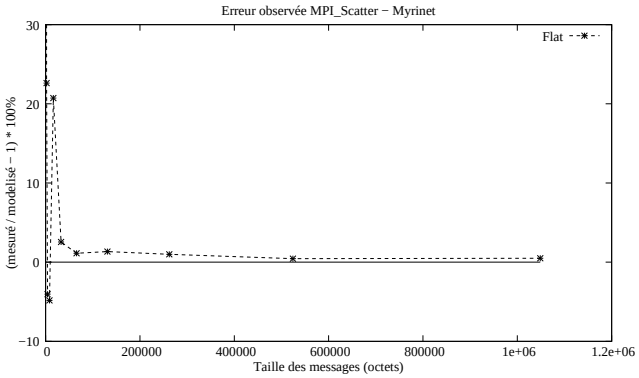


Figure 6.32 *Erreur des prédictions de l'opération MPI_Scatter pour l'envoi de messages de 1Mo*

6.4 Plusieurs vers Plusieurs : *MPI_Alltoall*

Un des plus importants patrons de communication collective pour des applications scientifiques est l'échange total [CHR99], où les algorithmes parallèles alternent des périodes de calcul avec des périodes d'échange de données entre les processus. Pour cela, une des opérations plus répandues est le *All-to-All*, qui permet la transposition des données appartenant à un groupe de processus. Dans le cas de l'opération *All-to-All*, chaque processus détient $m \times P$ unités de données qui seront distribuées également entre les P processus.

Plusieurs travaux visent l'optimisation de l'opération *All-to-All* et sa variante *All-to-All-v*, qui permet l'envoi des messages avec des tailles différentes pour chaque processus. Cependant, la plupart des propositions sont adaptées à des structures d'interconnexion très spécifiques, comme dans le cas des topologies en grille, tores et hypercubes [CAL95][CHR99][YAN00][KAL03]. Des solutions générales, comme celles implantées sur plusieurs distributions MPI, considèrent que chaque processus ouvre une communication directe avec les autres processus.

L'approche la plus simple d'implantation de *All-to-All*, que l'on appellera *Échange Direct*, considère que chaque processus communique directement avec les autres, et que tous les appels d'envoi et de réception sont initiés simultanément. Un exemple de l'approche *Échange Direct* est l'implantation de *MPI_Alltoall* de LAM version 6.5.2 [LAM03]. Dû à ses caractéristiques, cet algorithme peut avoir des problèmes de surcharge du récepteur, car les processus suivent le même ordre d'envoi, surchargeant un seul processus récepteur à chaque tour. À cause de cela, une optimisation simple consiste en faire la rotation des listes de destinataires, comme le font déjà les implantations MPI LAM 7.0.4 [LAM04] et MPICH 1.2.5 [MPI03]. Malgré cette optimisation, des tests pratiques effectués sur les réseaux Fast Ethernet, Giga Ethernet et Myrinet n'ont pas démontré une grande influence sur le résultat, comme démontrent les Figures 6.33, 6.34 et 6.35. Nos expériences suggèrent que la surcharge d'un récepteur est un problème mineur en comparaison avec l'occurrence de la congestion réseau. En fait, l'analyse faite par Grove [GRO03] indiquait déjà que les ralentissements observés sont plutôt dus à des pertes de paquets et leurs *timeouts* de retransmission TCP/IP causés par la surcharge du réseau.

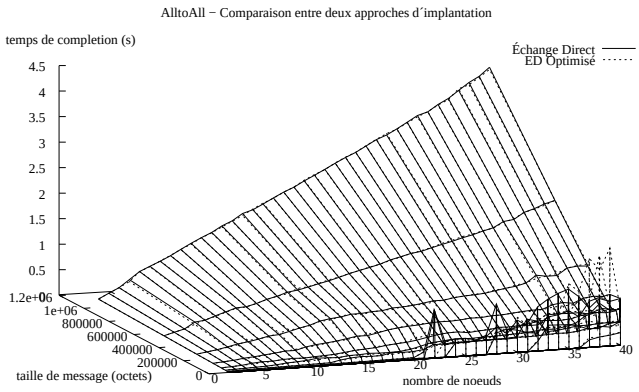


Figure 6.33 *Comparaison entre les approches* Échange Direct *et* Échange Direct Optimisé *pour un réseau Fast Ethernet*

Par conséquent, la difficulté des modèles de communication pour le *All-to-All* réside dans la prise en compte des spécificités du patron de communication *Plusieurs vers Plusieurs Personnalisé*. Des modèles théoriques comme ceux présentés par Christara [CHR99] et Pjesivac [PJE05] sont pour la plupart des simples extensions du modèle *Scatter*, qui ne tiennent pas compte de l'influence de la congestion réseau.

6.4.1 Congestion du réseau

Dans une opération de type All-to-All, chaque processus détient $m \times P$ items de données qui doivent être équitablement distribuées entre les P processus. Parce que les implantations de l'opération All-to-All utilisent généralement des communications point-à-point entre les processus, le réseau peut facilement devenir saturé, et par conséquent, dégrader la performance du réseau. En effet, Tam et Wang [TAM99][TAM01] ont démontré que le temps d'exécution des opérations de communication collective avec un fort échange de données est fortement dominé par la congestion

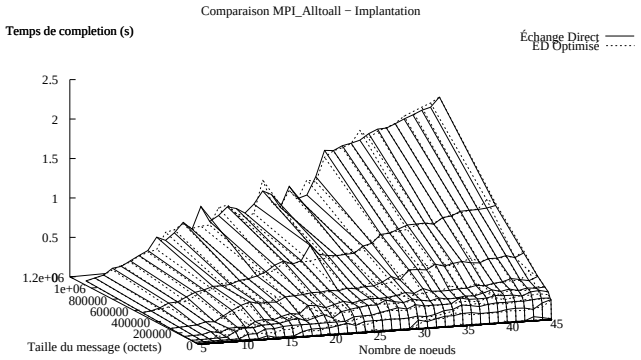


Figure 6.34 *Comparaison entre les approches* Échange Direct *et* Échange Direct Optimisé *pour un réseau Giga Ethernet*

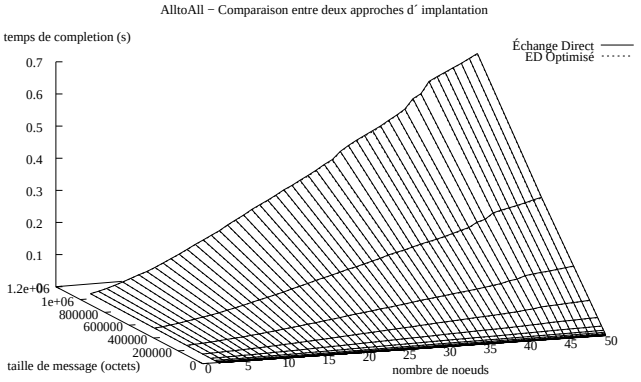


Figure 6.35 *Comparaison entre les approches* Échange Direct *et* Échange Direct Optimisé *pour un réseau Myrinet*

du réseau et par la perte de paquets de messages, ce qui rend très difficile la quantification de ces effets.

Malheureusement, la plupart des modèles de communication comme ceux présentés par [CHR99] ou Pjesivac-Grbovic [PJE05] sont des simples extensions du patron de communication *Un vers Plusieurs Personnalisé* (dont l'opération Scatter, où un simple processus envoie des différents messages de taille m à chaque autre processus). De tels modèles ne tiennent pas compte de l'influence de la congestion du réseau, et par conséquent, ne sont pas assez précis pour prédire la performance des opérations de type All-to-All.

Ce n'est que plus récemment que les modèles de communication tiennent en compte la congestion du réseau, comme indiqué par Grove [GRO03]. Un des premiers modèles qui considère les effets de la congestion des ressources a été présenté par Adve [ADV93]. Ce modèle considérait que le temps total d'exécution était réparti entre quatre composants :

$$T = t_{calcul} + t_{communication} + t_{contention} + t_{synchronisation}$$

Malgré sa simplicité conceptuelle, ce modèle n'est pas trivial à cause de la nature non-déterministe de la congestion, et surtout de la difficulté à déterminer les retards moyens de synchronisation.

Toutefois, des techniques furent suggérées pour corriger l'écart observé entre les prédictions et les valeurs mesurées. Bruck [BRU97b] suggéré l'utilisation d'un *facteur de ralentissement* (*slow-*

down factor en anglais) pour corriger les prédictions ; Clement *et al.* [CLE96] proposent l'utilisation d'un facteur de congestion γ qui augmente un modèle de communication linéaire T et qui permet l'expression de la congestion sur des réseaux partagés, comme par exemple l'Ethernet non commuté :

$$T = l + \frac{b\gamma}{W}$$

où l est la latence du lien, b est la taille du message, W est le débit du lien et γ représente le nombre de processus. Ce modèle augmente la précision des prédictions avec un coût minimal, mais pour cela il faut encore que tous les processus communiquent simultanément, ce qui n'est pas vrai que pour quelques patrons de communication. Néanmoins, dans les cas où cette hypothèse tient, ils ont observé que ce modèle de congestion a augmenté considérablement la précision des prédictions pour un coût presque nul.

Ce résultat est fortement lié au travail de Labarta et *al.* [LAB96], qui tente d'approcher le comportement de la congestion réseau en considérant que s'il y a m messages à transmettre, et seulement b canaux disponibles, les messages sont sérialisés en $\lceil \frac{m}{b} \rceil$ vagues de communication.

Certains modèles de performance orientés à la congestion sont apparus récemment. *LoGPC* [MOR01] est une extension du modèle *LogP* qui détermine l'influence de la congestion à travers l'analyse des filles d'attente sur un réseau de n cubes de dimension k chacun. Malheureusement, cette analyse rend très difficile l'utilisation pratique du modèle.

Une autre approche, plus pratique, est celle de Tam [TAM01], qui considère la congestion comme partie intégrante de la latence. Par conséquent, ce modèle utilise des valeurs de latence qui varient selon la taille du message. Si cette approche est beaucoup plus simple à implanter que celle de *LoGPC*, le modèle de Tam ne prend en compte que le nombre de processus communicants, paramètre qui est forcement liée à aux effets de congestion du réseau. De plus, le surcoût dû à l'obtention des valeurs de latence pour plusieurs tailles de messages devient trop élevé quand on considère des réseaux de longue distance.

6.4.2 Une approche paramétrique

Pour mieux évaluer l'influence de la congestion sur les connexions locales, nous avons réalisé une expérience similaire à celle déjà présenté en chapitre4 sur le débit agrégé d'un lien de longue distance.

Pour illustrer cet expérience, nous avons mesuré le débit moyen de la transmission d'un message de 32 Mo échangé entre plusieurs noeuds de la grappes d'Orsay (GdX). La Figure 6.36 indique en effet une brusque réduction du débit moyen, qui perd presque 50% de sa capacité quand le nombre de connexions simultanées augmente.

En effet, la Figure 6.37 indique qu'il existe un phénomène récurrent de perte de paquets à partir d'un nombre relativement réduit de connexions simultanées. Cette perte est identifié par l'augmentation du temps de transmission de certaines connexions, qui nécessitent beaucoup plus de temps pour compléter l'envoi du message de 32 Mo. Ce type de phénomène a déjà été étudié par Grove [GRO03], qui a établi un rapport entre le retard subit par certaines connexions et le temps nécessaire à la détection des pertes et la retransmission subséquente des paquets perdus.

L'occurrence de telles pertes de paquets a une conséquence majeure sur la modélisation des communication collectives. En effet, les opérations de communication collective dépendent de la complétion de toutes les transmissions, ce qui se traduit, spécialement dans le cas l'opération *All-to-All*, par une réduction importante de la performance de l'opération.

Ainsi, nous avons deux possibilités pour modéliser correctement la performance du *All-to-All* dans une condition pareille : acquérir les paramètres pLogP à partir d'un réseau saturé ou augmenter les modèles existants pour compenser le surcoût dû aux pertes de paquets. Alors qui d'un point de vue théorique la première approche semble la plus adaptée, elle présente deux désavantages :

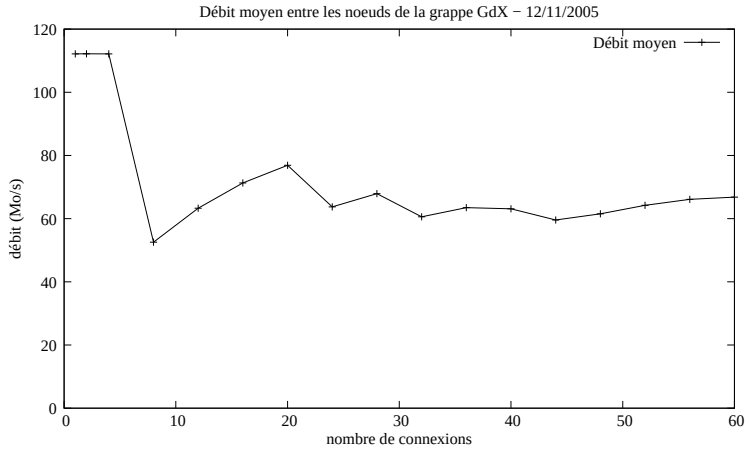


Figure 6.36 *Débit moyen de chaque connexion à l'intérieur de la grappe GdX*

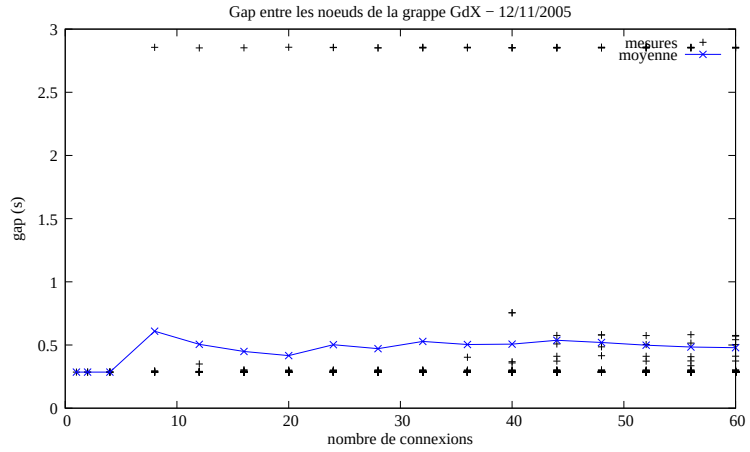


Figure 6.37 Gap *mesuré entre les noueds de la grappe GdX*

initialement, elle n'assure pas que l'ensemble des paramètres mesurés sera réellement influencé par la perte de paquets (une fois que la plupart des connexions ne subit pas des retards). Ensuite, cette approche implique une relecture des modèles établis pour d'autres opérations comme le *Broadcast*, qui jusqu'à ce moment correspondaient aux performances réelles des opérations.

À cause de ces deux aspects, nous adoptons dans ce travail une approche similaire à Clement et Steed [CLE96], où la congestion est suffisamment linéaire pour être modélisée. Notre approche consiste à identifier le comportement de l'opération *All-to-All* par rapport à la borne inférieure de communication établie à partir du modèle de communication *1-port*. Notre hypothèse est que la congestion dépend plutôt des caractéristiques physiques du réseau (cartes, liens, commutateurs, ...), de façon que le rapport entre le résultat pratique et la performance théorique devient une « signature » de ce réseau. Une fois identifié cette signature, nous pouvons l'utiliser pour prédire la performance d'autres exécutions effectuées sur le même réseau.

Pour cela, nous avons étudié le comportement des communications entre plusieurs interlocuteurs par rapport aux limitations du modèle de communication 1-*port*. Dans le modèle 1-*port*, un processus

ne peut pas envoyer des messages à deux processus simultanément ; cependant, un processus peut envoyer et recevoir des messages en même temps, si les deux interlocuteurs sont des processus différents [WAG04].

Initialement nous considérons les communications dans un environnement libre de congestion. Dans ce cas, un processus qui doit envoyer des messages de taille m à $P-1$ processus nécessite au moins $(P-1) \times g(m)$ unités de temps. Cependant, les messages reçus dans cette intervalle sont déjà disponibles grâce à la propriété du modèle *1-port*. Ainsi, même si chacun des P processus démarre ses communications simultanément avec les autres $P-1$ processus, le temps total d'émission et réception ne devrait pas dépasser $(P-1) \times g(m) + L$ unités de temps (équivalent au modèle Scatter utilisé par Christara [CHR99] et Pjesivac [PJE05]).

Cependant, plusieurs autres facteurs peuvent influencer le temps nécessaire pour la complétion de l'opération MPI_Alltoall. Mis à part le décalage existant entre l'instant où chaque processus démarre ses communications, l'opération MPI_Alltoall a la tendance à saturer les liens de communications, et par conséquent, elle est notamment sensible à la congestion. Dans ce cas, le modèle Scatter n'est plus adéquat, une fois qu'il n'est pas capable de représenter les effets de la congestion.

Toutefois, la connaissance de la borne inférieure peut aider à évaluer la performance réelle : elle permet la quantification des facteurs liés à la congestion du réseau, et la relation entre la valeur théorique et la valeur réelle devient une caractéristique intrinsèque du réseau étudié, une « signature » de ce réseau.

Notre approche pour modéliser la performance de l'opération MPI_Alltoall malgré l'influence de la congestion réseau consiste en déterminer une *relation de congestion* γ qui fait la relation entre la limite théorique et la performance réelle mesurée. Par simplicité, nous considérons que cette *relation de congestion* γ est constante et dépend uniquement des caractéristiques du réseau. Cette relation de congestion est paramétrée par rapport au nombre de processus communicants et à la taille des messages échangés, une fois que les limites théoriques dépendent de ces valeurs.

L'utilisation de la *relation de congestion* γ permet déjà une bonne approximation des performances réelles, comme attesté par [BAR04c, BAR05a]. Toutefois, nous avons constaté que la performance des communications dans certains réseaux était sujette à d'autres facteurs qui l'éloignaient des prédictions. Un exemple de ces variations est observé dans la Figure 6.38, où les temps de communication mesurés subissent une forte augmentation à partir d'un certain moment. En effet, cette augmentation du temps de communication semble se produire quand le volume de données échangées dépasse les 64Ko ; pour cette raison, nous croyons que ce phénomène est lié au changement de politique de transmission de la bibliothèque MPI utilisée, ou même à la capacité des buffers de communication des machines et du commutateur.

Pour mieux représenter les effets de cette surcharge des buffers de communication, nous avons ajouté à notre modèle un facteur supplémentaire δ_P. Ce facteur permet la représentation des coûts dus à la surcharge des buffers, mais aussi ceux dus au surcoût de la segmentation des messages. Ce facteur δ_P dépende du nombre de processus communicants et de la taille des messages envoyés.

Avec l'adjonction du facteur δ_P, un modèle de performance réaliste pour l'opération MPI_Alltoall est défini selon l'expression suivante :

$$\begin{aligned} T &= SansCongestion \times \gamma + (P-1) \times \delta_P \\ &= ((P-1) \times g(m) + L) \times \gamma + (P-1) \times \delta_P \\ &= (P-1) \times (g(m) \times \gamma + \delta_P) + L \times \gamma \end{aligned}$$

6.4.3 Définition des paramètres

Pour illustrer notre approche, nous présentons dans cette section la procédure utilisée pour définir les paramètres γ et δ_P à partir des expériences pratiques effectuées avec l'algorithme *Échange Direct* sur les réseaux Fast Ethernet, Giga Ethernet et Myrinet.

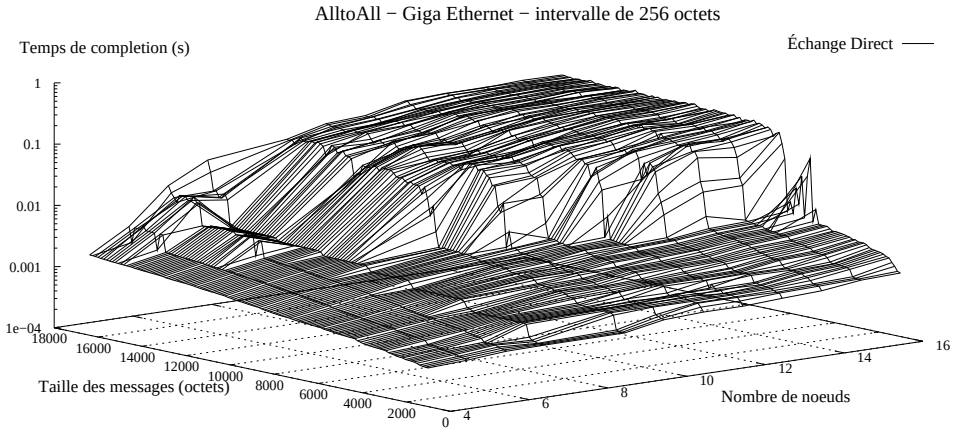


Figure 6.38 *Détail du comportement de l'opération All-to-All*

Comme expliqué précédemment, notre approche consiste à comparer la performance réelle et les prévisions théoriques obtenues pour un groupe donné de machines ; ces données nous permettent l'adéquation des paramètres γ et δ_P, qui seront ensuite utilisés comme une « signature » des réseaux, de manière à prédire la performance de l'opération All-to-All sous un nombre variable de processus.

Les paramètres γ et δ_P sont obtenus par régression linéaire, avec la méthode du moindre carré. Pour cela, nous utilisons au moins quatre points de mesure, notamment avec des messages d'une taille supérieure à 100 Ko ; ce choix est dû à la présence d'instabilités pour les petits messages.

Réseau Fast Ethernet

Dans le cas du réseau Fast Ethernet, la performance mesurée est légèrement supérieure à la limite inférieure ; ce petit écart s'explique surtout par le débit de connexion de ce réseau, assez modéré pour minimiser la surcharge des liens. Il indique toutefois qu'il y a un coût d'initialisation qui n'est pas normalement pris en compte lors de la modélisation du All-to-All.

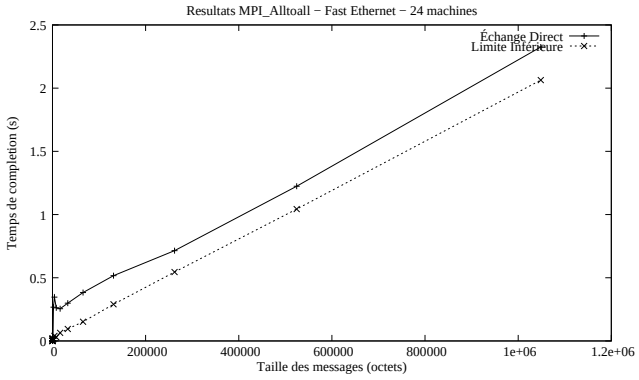


Figure 6.39 *Limites théoriques et performance mesurée dans un réseau Fast Ethernet*

À partir des observations réelles et des prédictions des limites théoriques illustrées par les figures 6.39, nous avons pu établir une *relation de congestion* γ tel que les prédictions soient proches des résultats réels. Pour cela, nous avons identifié une valeur de $\gamma = 1,0195$ pour le réseau Fast Ethernet. À cette valeur s'ajoute un surcoût supplémentaire δ_P pour les messages plus grands que 2Ko,

très probablement à cause de la surcharge des buffers de réception et de la segmentation implicite des messages en plusieurs paquets Ethernet. Ce surcoût est proportionnel au nombre de processus communicants, et dans le cas du réseau Fast Ethernet évalué, nous avons trouvé $\delta_P = 8,23\,ms$, de manière à permettre une approximation de la performance réelle comme illustré dans la Figure 6.40.

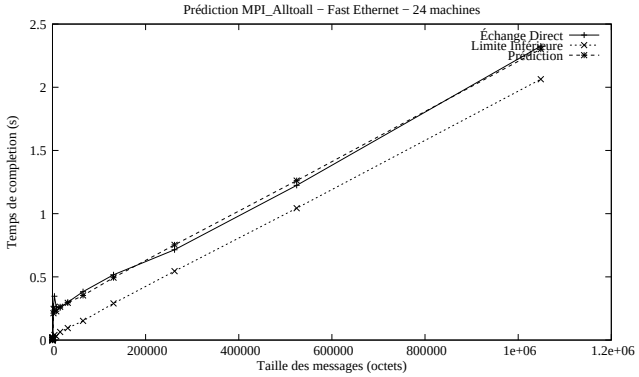


Figure 6.40 *Approximation du résultat réel*

La différence entre les prédictions et les valeurs mesurées est généralement inférieure à 10% comme atteste la Figure 6.41.

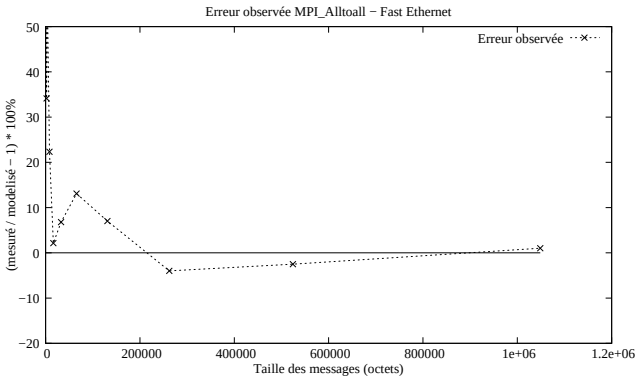


Figure 6.41 *Erreur des prédictions pour un groupe de 24 machines*

Réseau Giga Ethernet

Dans le cas du réseau Giga Ethernet, la performance mesurée est nettement supérieure à la borne inférieure. Le débit de connexion plus élevé de ce réseau augmente les chances de saturer les liens et le commutateur. Ce dépassement est tout à fait normal, car que la limite théorique n'est qu'une estimation fondée sur un scénario de communication très simples et optimiste.

À partir des observations réelles et des prédictions pour la borne inférieure illustrées par les figures 6.42, nous avons pu établir des *relations de congestion* γ tel que les prédictions soient proches des résultats réels. Pour cela, nous avons identifié une valeur de $\gamma = 4,3628$ pour le réseau Giga Ethernet. À cette valeur s'ajoute un surcoût supplémentaire δ_P pour les messages plus grands que 8Ko, à cause de la surcharge des buffers de réception et de la segmentation des messages en paquets Ethernet. Ce surcoût est proportionnel au nombre de processus communicants, et dans le cas du réseau Giga Ethernet évalué, nous avons trouvé $\delta_P = 4,93\,ms$, de manière à permettre une approximation de la performance réelle comme illustré dans la Figure 6.43.

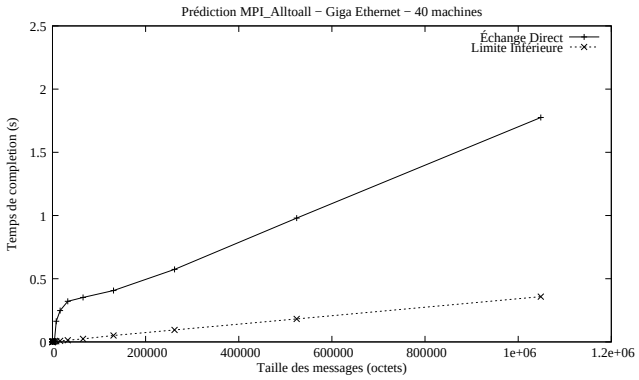


Figure 6.42 *Limites théoriques et performance mesurée dans un réseau Giga Ethernet*

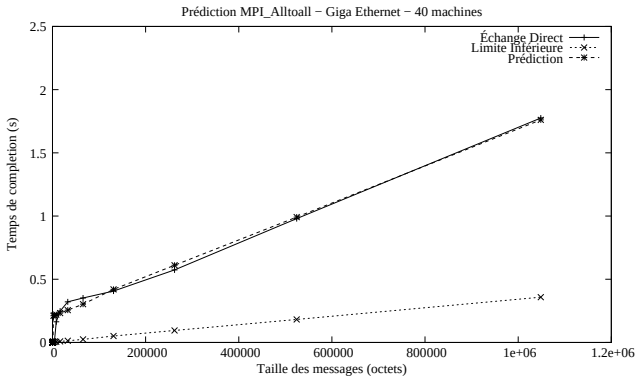


Figure 6.43 *Approximation du résultat réel*

À l'exemple de ce qui a déjà été constaté avec l'opération MPI_Bcast, certains noeuds ont un comportement plus instable. Cependant, nous pouvons encore vérifier que la différence entre les prédictions et les valeurs mesurées n'est pas très importante ; en effet, l'erreur s'avère très réduite quand les processus sont plus stables, comme atteste la Figure 6.44. Toutefois, des erreurs plus prononcées sont observées dans le cas des petits messages, à l'exemple du réseau Fast Ethernet.

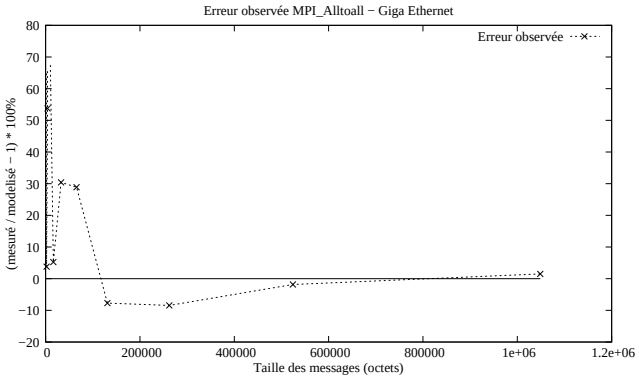


Figure 6.44 *Erreur des prédictions pour un groupe de 40 machines*

Réseau Myrinet

Dans le cas du réseau Myrinet, la performance mesurée est, à l'exemple du réseau Giga Ethernet, bien supérieure à la borne inférieure ; cet écart indique que dans cette architecture réseau la surcharge due à la congestion du réseau peut représenter un facteur très important.

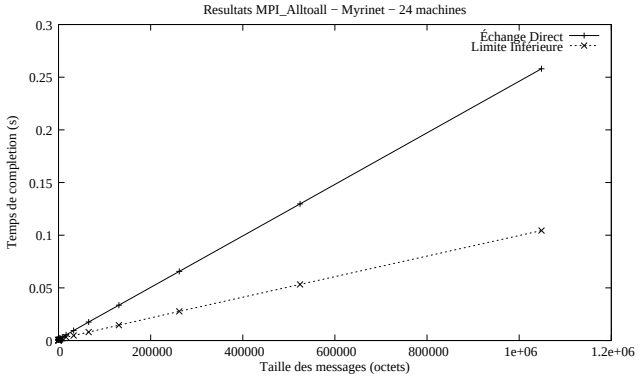


Figure 6.45 *Limites théoriques et performance mesurée dans un réseau Myrinet*

À partir des observations réelles et des prédictions des limites théoriques illustrées par les figures 6.45, nous avons pu établir des *relations de congestion* γ tel que les prédictions soient proches des résultats réels. Pour cela, nous avons identifié une valeur de $\gamma = 2,49754$ pour le réseau Myrinet. Cependant, nous n'avons pas constaté un surcoût δ_P supplémentaire (la régression linéaire indique une valeur de moins de 1 microseconde), de manière à ce que l'approximation de la performance réelle illustrée dans la Figure 6.46 est obtenue seulement avec le paramètre γ.

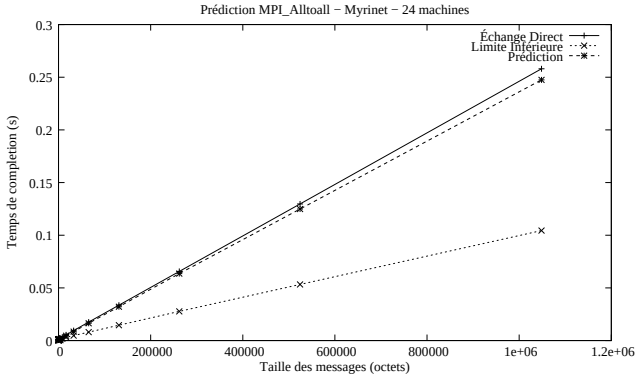


Figure 6.46 *Approximation du résultat réel*

La différence relative entre les prédictions et les valeurs mesurées, présentée dans la Figure 6.47, est très réduite pour les messages plus grands, et représente quelques millisecondes seulement dans le cas des petits messages. De même, nous n'observons pas des retards importants comme ceux observés sur les réseaux Fast Ethernet et Giga Ethernet, une fois que les expériences utilisent le protocole propriétaire du réseau Myrinet, qui ne subit pas les mêmes problèmes de l'implantation du protocole TCP.

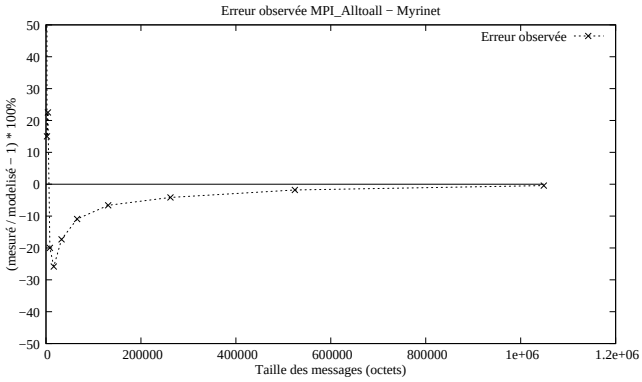


Figure 6.47 *Erreur des prédictions pour un groupe de 24 machines*

TAB. 6.4 *Paramètres pour la prédiction de performance du All-to-All*

	γ	δ_P
Fast Ethernet	$1,0195$	$8,23\,ms$
Giga Ethernet	$4,3628$	$4,93\,ms$
Myrinet	$2,4975$	0

6.4.4 Validation des modèles

Une fois que la *signature* γ du réseau et le surcoût δ_P ont été identifiés, nous avons appliqué ces paramètres, regroupés dans le Tableau 6.4 à la prédiction des performances d'un nombre arbitraire de noeuds.

Nous pouvons observer que ces paramètres permettent une estimation de la performance des opérations de type All-to-All avec une grande précision. En effet, les Figures 6.48, 6.49 et 6.50 présentent la comparaison entre les prédictions de performance et les valeurs mesurées pour les réseaux Fast Ethernet, Giga Ethernet et Myrinet, respectivement.

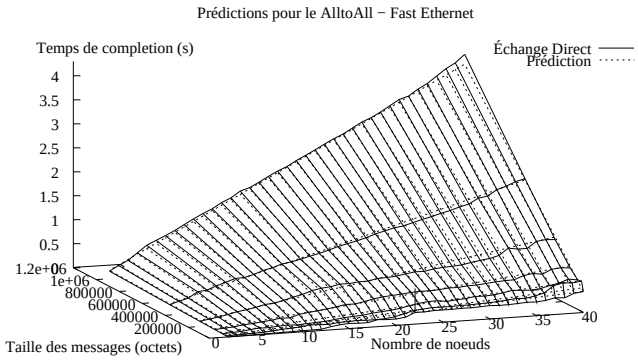


Figure 6.48 *Prédictions de performance du All-to-All pour un réseau Fast Ethernet*

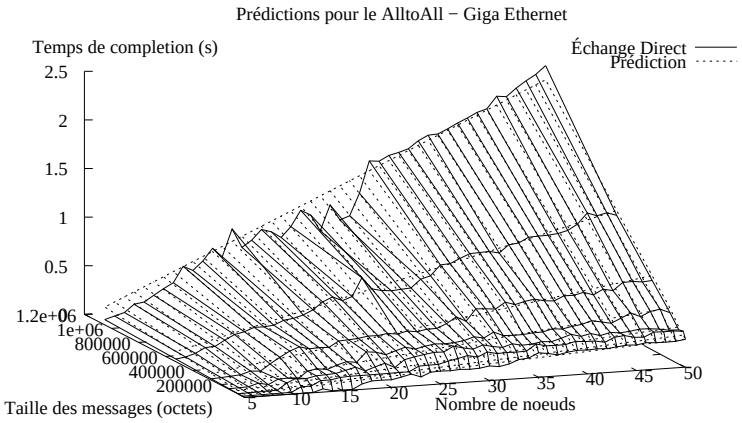


Figure 6.49 *Prédictions de performance du All-to-All pour un réseau Giga Ethernet*

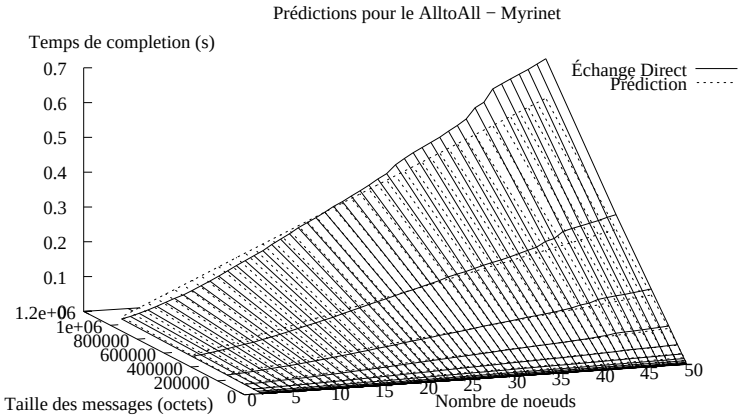


Figure 6.50 *Prédictions de performance du All-to-All pour un réseau Myrinet*

6.5 Discussion

Des travaux récents visent l'optimisation des opérations de communication collective dans les environnements de type grille de calcul. La solution la plus répandue est la séparation des communications internes et externes à chaque grappe, comme le font les systèmes ECO [LOW96], MagPIe [KIE99b][KIE01], et LAM-MPI 7 [LAM04], mais cela n'exclut pas le découpage des communications en plusieurs couches, pratique efficace démontrée par Karonis *et al.* [KAR00]. Dans les deux cas, la prédiction des performances est un facteur essentiel, soit pour le réglage fin des paramètres de communication, soit pour le calcul de la distribution et de la hiérarchie des communications. Pour cela, il est très important d'avoir des modèles précis des communications collectives, lesquels seront utilisés pour prédire ces performances.

Ce chapitre a présenté notre expérience dans le domaine de la modélisation des opérations de communication collective. L'efficacité de ces modèles est analysée à travers la comparaison entre les prédictions de performance et les résultats réels obtenus pour trois importants patrons de communication collective : *Un vers Plusieurs*, *Un vers Plusieurs Personnalisé* et *Plusieurs vers Plusieurs Personnalisé*. Pour cela, les expériences ont utilisé trois architectures réseaux distinctes, Fast Ethernet, Giga Ethernet et Myrinet. Nous démontrons que les modèles de communication sont suffisamment précis pour prédire les performances de ces opérations collectives sur les trois environnements réseaux étudiés, et aussi pour permettre la sélection des techniques les plus adaptées à chaque situa-

tion.

Une contribution importante de ce travail est l'effort pour modéliser les opérations de type *Plusieurs vers Plusieurs Personnalisé*. En général, ces opérations sont sujettes à des retards importants dus aux effets de la congestion du réseau. Dans notre approche, les facteurs de congestion affines $\gamma + \delta_P$, obtenus à partir des modèles de performance théoriques, sont utilisés pour prédire les performances de ce type d'opération collective avec une bonne précision et surtout un coût très bas. Même si notre modèle de communication *Plusieurs vers Plusieurs Personnalisé* ne couvre pas tous les effets de congestion qui peuvent influencer les résultats réels, en particulier dans le cas des petits messages, il fournit des indices qui contribuent à la recherche de modèles plus précis.

III

Communication Collective dans les Environnements de type Grille de Calcul

7

Communications sur grilles

Plusieurs travaux récents visent l'implantation des opérations de communication collective adaptées aux systèmes à grande échelle, notamment les grilles. Dans ces environnements, l'hétérogénéité est un facteur prépondérant qui doit obligatoirement être pris en compte, comme nous l'avons déjà observé [BAR04b]. Cette hétérogénéité représente néanmoins un vrai défi pour la prédiction des performances, car les facteurs qui influencent les communications ont des origines très variées, comme la distribution des processus (par exemple, sur une grappe de machines multiprocesseurs), la distance entre les machines et/ou les grappes, le taux d'utilisation du matériel (surtout la congestion du réseau) et la variation de performance du matériel. En effet, les grilles de calcul combinent, la plupart du temps, différentes machines et réseaux.

L'hétérogénéité inhérente à ces environnements, associée à la volatilité des noeuds dans les grilles de calcul, empêche la création d'opérations spécifiques pour ces environnements, comme en attestent [BHA03] et [VAD00]. Pour simplifier cette modélisation, la plupart des solutions considèrent les grilles comme l'interconnexion d'îlots de grappes homogènes. Dans ce contexte, la majorité des systèmes concentre l'optimisation au niveau des communications entre les grappes, puisque ces liaisons sont généralement plus lentes que celles intérieures à la grappe. Quelques exemples de cette approche en deux couches incluent les bibliothèques ECO [LOW96], MagPIe [KIE99b][KIE01], et même la bibliothèque LAM-MPI 7 [LAM04], qui considère les machines SMP comme des îlots de communication rapide. Il reste néanmoins la nécessité de régler les paramètres de communication pour avoir des performances optimales. Pour cela, la prédiction des performances à travers des modèles de communications est un choix très avantageux.

Il existe toutefois la possibilité d'organiser les communications en un plus grand nombre de couches. En effet, le travail de Karonis et *al.* [KAR00][KAR02] a démontré que le découpage en plusieurs couches de communication peut conduire à des réductions du temps d'exécution plus importantes qu'un découpage en deux couches. Cependant, il est nécessaire de connaître *a priori* le coût de communication interne à chaque grappe. Dans ce cas, le calcul de la distribution et de la hiérarchie des communications dépend des temps de communication à l'intérieur des grappes. Ces temps varient selon l'opération de communication collective, le nombre de noeuds et les caractéristiques du réseau de chaque grappe.

7.1 Broadcast

7.1.1 État de l'art

L'opération *Broadcast* est une des plus simples opérations de communication collective : initialement, seul le processus *racine* détient le message qui doit être diffusé ; à la fin de l'opération, une copie de ce message est déposée dans chaque processus du groupe. La détermination du meilleur arbre de diffusion pour un environnement homogène est une tâche relativement facile : le choix de l'arbre dépend surtout des paramètres d'interconnexion.

Cependant, dans le cas d'un réseau hétérogène, ce problème devient bien plus difficile : en effet, l'identification du meilleur arbre de diffusion est un problème NP-complet [BHA99][BEA04c, BEA05b][WUJ04].

En raison de ces restrictions, plusieurs travaux s'intéressent au développement de techniques d'approximation qui soient suffisamment efficaces pour être utilisées dans un système réel. Parmi ces travaux on note celui de Beaumont *et al.* [BEA04c, BEA05b], Banikazemi [BAN98], Bhat [BHA99, BHA03], Liu [Liu03], Park [PAJ96], Mateescu [MAT05], Miled [MIL98] et Vorakosit [VOR03]. L'approche de Beaumont utilise des algorithmes de type « *prunning* » pour identifier le meilleur arbre de diffusion possible à l'intérieur d'un réseau, alors que la plupart des travaux font la construction d'un arbre de diffusion à partir d'une source donnée. L'approche *« prunning »* est plus adaptée à une diffusion en régime continu, comme par exemple la transmission d'un flux multimédia. Leurs efforts sont alors concentrés sur l'optimisation du débit maximal atteint.

L'approche la plus générale, utilisée par les autres auteurs cités ci-dessus (Banikazemi, Bhat, Vorakosit, etc.), utilise comme processus racine un processus indiqué par l'utilisateur. Cette approche est plutôt orientée vers des diffusions occasionnelles où la source du broadcast peut changer d'un appel à l'autre. Ce modèle correspond plus exactement au format de l'appel MPI_Bcast.

La plupart des travaux dédiés à l'optimisation des communications collectives dans des environnements hétérogènes font référence aux processus communicants. C'est-à-dire, les optimisations sont faites en tenant compte du coût d'interconnexion entre chaque paire de processus concernée par le broadcast. C'est le cas de Banikazemi [BAN98], Bhat [BHA99, BHA03] et Mateescu [MAT05], qui utilisent différentes stratégies pour construire des arbres de diffusion optimisés.

Cependant, l'environnement de type grille est normalement caractérisé par un grand nombre de processus communicants, résultat de l'association des différentes grappes de calcul. Dans ce cas, la complexité de la tâche d'optimisation est bien plus importante, et des simplifications s'imposent afin de permettre l'utilisation de telles méthodes dans la pratique. Une de ces simplifications est le regroupement des processus selon leurs performances relatives (par exemple, par rapport à la communication), de manière à ce que toute une classe de processus puisse être traitée comme une entité unique. De cette manière, l'augmentation massive du nombre de processus dans une grille de calcul peut être encore facilement abordable par les méthodes d'optimisation classiques.

Toutefois, cette simplification au niveau de l'optimisation requiert en contrepartie l'organisation des différentes classes de processus autour d'un ou plusieurs « coordinateurs ». Ces derniers sont des processus responsables de la communication avec les autres grappes et de la diffusion à l'intérieur du groupe de processus. Cette organisation des processus assume tout naturellement une disposition hiérarchique, où les messages sont diffusés d'abord entre les coordinateurs des groupes, et finalement vers les autres processus.

L'un des premiers travaux à adresser ce problème pour les grilles est celui de Lowekamp [LOW96, LOW00], où les machines sont regroupées selon leur localisation dans le réseau. Le même principe est suivi par la bibliothèque MagPIe [KIE99b], de manière à ce que les échanges de messages à travers les liens plus lents (ceux qui connectent les différentes grappes) soient réduits au minimum.

Une caractéristique commune à ces deux travaux est que la communication se fait en deux couches, l'*inter-grappes* et l'*intra-grappes*. Si cette disposition permet la minimisation du trafic sur les liens les plus lents, elle ne tire pas partie des caractéristiques du Broadcast comme la multi-

plication des sources de diffusion, principe de base du Broadcast en systèmes homogènes. Karonis [KAR00, KAR02] a adressé ce problème en établissant des communications à plusieurs niveaux, de manière à permettre le recouvrement des communications des différentes grappes et à minimiser le temps de diffusion inter-grappes. Cependant, l'approche de Karonis est purement orientée par la latence de communication, et comme l'atteste Lacour [LAC01], l'implantation de MPICH-G2 identifie les différents niveaux selon leurs latences relatives (cf. Tableau 7.1). De cette manière, la communication entre différentes grappes (niveaux 0 et 1) est établie de façon similaire au modèle en deux couches, ce qui ne permet pas une véritable amélioration par rapport au modèle de MagPIe.

TAB. 7.1 *Ordre des niveaux selon la latence des protocoles de communication [LAC01]*

Niveau 0 >	Niveau 1 >	Niveau 2 >	Niveau 3, 4, ...
WAN-TCP	LAN-TCP	localhost-TCP	mémoire partagée Myrinet MPI fabricants

Cependant, à défaut de leur apport aux algorithmes de Broadcast, ces techniques peuvent être améliorées. En effet, les travaux précédents ont été établis dans un contexte où les communications de longue distance étaient plusieurs ordres plus lentes que celles à l'intérieur des réseaux locaux, et la réduction des communications inter-grappes permettait la minimisation de la congestion sur les liens les plus lents. Si cela est encore vrai en ce qui concerne la latence entre les noeuds, il n'est plus exact pour le débit d'un lien de longue distance. D'autre part, le faible coût du matériel informatique permet aujourd'hui que les grappes regroupent des centaines de noeuds. Or, plus le coût de diffusion « intra-grappes » devient important, plus son influence sur la performance sera importante au moment de définir l'ordonnancement des communications.

C'est exactement ce qui différencie les heuristiques traitant (ou ne traitant pas) la communication à l'intérieur des groupes : les heuristiques « traditionnelles » et celle appelées « heuristiques sensibles au contexte des grilles de calcul » (« *grid-aware* » en anglais). Dans le premier cas, l'optimisation ne tient compte que des communications entre les différents coordinateurs, alors que le deuxième cas s'occupe aussi de la diffusion à l'intérieur des grappes.

Afin de permettre une optimisation moins centrée sur la latence, et qui fait aussi usage des autres paramètres de connexion dont nous disposons, nous avons étudié certaines heuristiques proposées par Banikazemi [BAN98], Bhat [BHA03] et Liu [Liu02], proposées initialement dans le cadre des réseaux hétérogènes. Ces heuristiques ont pour objectif la construction d'arbres de diffusion (à travers l'ordonnancement des communications) qui minimisent le temps de complétion du Broadcast. Alors que ces heuristiques peuvent être utilisées pour la construction d'un arbre de diffusion complet, cela implique une augmentation de la complexité des optimisations qui parfois n'est pas justifiée, comme par exemple dans le cas des réseaux homogènes, dont nous connaissons déjà la meilleure stratégie de diffusion (conformément à ce qui a été présenté en chapitre 6). C'est pour cette raison que nous avons utilisé ces heuristiques seulement pour structurer la communication inter-grappes, intrinsèquement hétérogène.

Les prochaines sections présentent les différentes heuristiques étudiées pour l'optimisation des communications de type MPI_Bcast. Certaines de ces heuristiques sont la simple application des méthodes pour les réseaux hétérogènes dans le contexte des grappes hiérarchisées. Dans ce travail nous proposons trois nouvelles méthodes, qui au contraire des techniques précédentes, considèrent autant les communications entre les coordinateurs que les temps nécessaires à la diffusion des messages à l'intérieur des grappes.

Formalisme utilisé

Pour décrire les heuristiques présentées dans cette section, nous utilisons un formalisme de groupes similaire à celui de Bhat [BHA03]. Dans ce formalisme, les grappes sont séparées en deux groupes, **A** et **B**. Le groupe **A** contient les grappes qui ont déjà reçu le message (la réception du message par le coordinateur de la grappe est suffisante). Le groupe **B** contient les grappes qui devront recevoir le message. De cette manière, le groupe **A** contient initialement la grappe du processus *source* ou *racine*, tandis que le groupe **B** contient toutes les autres grappes du réseau.

À chaque étape, un émetteur appartenant au groupe **A** et un récepteur appartenant au groupe **B** sont choisis. Après la communication entre ces deux grappes (plus exactement, leurs coordinateurs), la grappe réceptrice est transférée au groupe **A**.

L'implantation de ces communications est faite de manière à rendre prioritaires les communications entre les grappes. En effet, les coordinateurs diffusent le message à l'intérieur de ses grappes seulement après la fin des communications inter-grappes. Cette stratégie favorise la multiplication des sources disponibles et l'application des heuristiques, ainsi que la prédiction du temps total d'exécution du Broadcast.

7.1.2 Stratégies traditionnelles

7.1.2.1 Diffusion en Arbre Plat (Flat)

L'heuristique en *Arbre Plat* (approche utilisée par la bibliothèque MagPIe), découpe la communication en deux niveaux, *inter-grappes* et *intra-grappes*.

Dans le premier niveau, le processus *racine* envoie le message à tous les coordinateurs des différentes grappes. L'ordre d'envoi suit le « rang » des différentes grappes, prédéfini à l'initialisation de MagPIe (normalement, à travers le fichier *magpie_clusters*). Formellement, cela veut dire qu'à chaque étape, le processus *racine* choisi comme récepteur la première grappe du groupe **B**. Dans cette « heuristique », le processus émetteur est toujours le même (le processus *racine*), malgré le fait que les grappes qui ont déjà reçu le message font désormais partie du groupe **A**. Dans le deuxième niveau de diffusion, exécuté à l'intérieur de chaque grappe, les coordinateurs exécutent un *broadcast* en arbre binomial.

Même si cette heuristique est très simple à implanter, elle est toutefois très peu optimisée. En effet, la diffusion des données ne tient pas compte des performances des différentes grappes, ni les vitesses d'interconnexion entre les *coordinateurs*. Même si l'utilisateur organise le fichier de description des grappes de manière à favoriser les communications émises d'une certaine grappe, celles-ci restent soumises à une structure de diffusion *plate*.

Variante

À partir de l'approche adoptée par MagPIe, nous avons élaboré une variante légèrement plus adaptée aux différentes grappes. Pour cela, nous avons simplement recherché la stratégie de communication la mieux adaptée à l'exécution du deuxième niveau de diffusion à l'intérieur de chaque grappe, selon les modèles de communication présentés dans le chapitre 6.

Les apports de cette variation sont limités, ayant comme principal effet la réduction du temps de communication à l'intérieur des grappes les plus lentes. Toutefois, cela peut aider dans le cas où les grappes présentent de grandes variations de performance, ou lorsque les réseaux de communication favorisent d'autres stratégies de communication que l'arbre binomial.

7.1.2.2 Fastest Node First - FNF

L'heuristique *Fastest Node First* (le noeud le plus rapide d'abord) a été proposée par Banikazemi *et al.* [BAN98]. Dans leur modèle de communication, le réseau est composé d'un certain nombre

de noeuds P. À chaque noeud P_i on associe un coût d'envoi C_i. Ce coût C_i est indépendant de la destination et de la taille du message, et indique seulement la différence de vitesse entre les noeuds.

L'heuristique proposée par Banikazemi *et al.* nécessite $P-1$ itérations, où à chaque étape l'heuristique définie un émetteur et un récepteur. Le récepteur est choisi parmi les possibles récepteurs du groupe **B** dont le coût C_i est le plus petit. L'émetteur est le processus du groupe **A** qui peut finir la communication le plus rapidement possible. Cela dit, cette stratégie choisit l'émetteur le plus rapide et le récepteur qui pourrait retransmettre les messages le plus rapidement possible, à son tour.

L'efficacité de l'heuristique FNF dans le cadre des environnements homogènes a été démontrée par Liu [Liu00b], qui a prouvé que l'heuristique FNF produit des ordonnancements avec au plus deux fois le temps optimal.

Cependant, des environnements homogènes comme ceux considérés par Banikazemi sont assez rares dans les grilles, ce qui rend cette heuristique très limitée par rapport à la modélisation des communications. En effet, le modèle de coût unique C_i n'est pas suffisant pour représenter l'hétérogénéité d'un réseau d'interconnexion, comme indiqué par Bhat [BHA03].

7.1.2.3 Fastest Edge First - FEF

Proposée par Bhat *et al.* [BHA03], l'heuristique *Fastest Edge First* (l'arête la plus rapide d'abord) est un algorithme glouton qui fait partie d'une collection d'heuristiques proposées comme alternative à l'heuristique FNF.

Assez simple, cette heuristique est très similaire à l'heuristique FNF. Seulement, au lieu d'un coût de communication unique C_i, l'heuristique évalue le poids de chaque lien de communication $L_{i,j}$ entre deux noeuds différents (les arêtes), correspondants à la latence de communication entre les deux processus.

Pour identifier l'ordonnancement des communications nécessaire à l'exécution de l'opération de Broadcast, l'algorithme FEF, ordonne les processus du groupe A selon leurs arêtes les plus rapides. Cela permet le choix du lien le plus rapide parmi toutes les possibilités, et en même temps, sert à définir l'émetteur et le récepteur, déterminés implicitement par l'arête choisie. Une fois que le récepteur est désigné, celui-ci est transféré du groupe **B** vers le groupe **A**. À cet instant, les arêtes minimales doivent être recalculées.

Le raisonnement de cette heuristique est que le choix des liens les plus rapides permet d'augmenter rapidement le nombre d'émetteurs. À leur tour, ces émetteurs pourront disséminer le message vers les processus les plus éloignés, tout en choisissant le lien le moins coûteux.

7.1.2.4 Early Completion Edge First - ECEF

Selon les heuristiques précédentes, une fois que le récepteur était assigné, celui-ci était immédiatement transféré vers le groupe des émetteurs, le groupe **A**. Toutefois, à cause des délais de communication, il est possible que ce récepteur n'ait pas encore reçu le message et qu'il soit choisi pour le retransmettre à un deuxième processus. La communication subira un retard supplémentaire, alors qu'une autre arête, moins rapide, pourrait finir la transmission plus vite si son émetteur a déjà le message.

Pour tenir compte des retards dus à la transmission des données, l'heuristique *Early Completion Edge First* (arête qui finit le plus tôt) considère aussi dans son évaluation l'instant où les émetteurs ont les données disponibles pour l'envoi. Ainsi, la *disponibilité* RT_i du processus émetteur (*Ready Time* en anglais) est alors utilisée conjointement avec le temps nécessaire à la transmission du message entre les processus (le gap plus la latence) de manière à choisir le couple émetteur-récepteur qui minimise le temps :

$$RT_i + g_{i,j}(m) + L_{i,j}$$

Ainsi, l'objectif de cette heuristique est d'augmenter le nombre de processus qui peuvent *effectivement* transmettre les messages aux autres processus.

7.1.2.5 Early Completion Edge First with lookahead - ECEF-LA

Pour augmenter l'efficacité de l'heuristique précédente, la dernière heuristique proposée par Bhat *et al.* [BHA03] propose une recherche plus approfondie sur les possibles choix. En effet, si l'objectif des heuristiques précédentes était la multiplication des sources disponibles, cela suppose que ces sources pourront, à leur tour, retransmettre les messages de manière efficace.

C'est ainsi que Bhat a proposé l'utilisation d'une fonction de *lookahead* (recherche en avant) pour évaluer si le choix d'un récepteur est réellement bon. De cette manière, l'algorithme calcule préalablement la fonction de *lookahead* F_j pour tous les processus dans le groupe **B**, et la paire émetteur-récepteur est celle qui minimise la somme :

$$RT_i + g_{i,j}(m) + L_{i,j} + F_j$$

Cette fonction de *lookahead* peut être définie de plusieurs façons. Bhat [BHA03] propose, par exemple, le coût minimal pour que le processus j transmette à d'autres processus encore dans le groupe **B**. Cette fonction est alors la suivante :

$$F_j = \min_{P_k \in B} (g_{j,k}(m) + L_{j,k})$$

Intuitivement, cette fonction indique l'utilité du processus P_j si à son tour il est transféré vers le groupe **A**. D'ailleurs, Bhat a proposé d'autres fonctions de *lookahead*, dont par exemple la moyenne de la latence entre P_j et les autres processus en **B**, ou alors la latence moyenne entre les émetteurs et les récepteurs, si on considère que P_j est transféré vers le groupe **A**.

7.1.3 Heuristiques « sensibles au contexte des grilles de calcul »

Si parmi les heuristiques présentées précédemment nous avons déjà hiérarchisé les communications en deux niveaux - *inter-grappes* et *intra-grappes* - , jusqu'à présent les fonctions d'évaluation ne tiennent compte que des coûts de transmission entre les coordinateurs des différentes grappes du réseau.

Cependant, comme nous l'avons déjà exposé, le coût d'une communication hiérarchique ne dépend pas seulement des latences entre les différentes grappes, mais aussi du temps nécessaire à la diffusion des messages à l'intérieur de ces grappes. Ce coût de diffusion intra-grappes devient encore plus important avec l'augmentation du nombre de noeuds à l'intérieur des grappes, qui aujourd'hui dépasse facilement la centaine de machines. Par exemple, l'envoi d'un message de 1Mo entre les grappes *Grid eXplorer* et *icluster-2* requiert 349 millisecondes, alors que le broadcast de ce même message entre 50 noeuds de la grappe *icluster-2* peut nécessiter jusqu'à 3 secondes selon le réseau et l'algorithme utilisé. Si ce temps n'est pas pris en compte lors de la modélisation des communications, l'ordonnancement des communications risque d'être sous-optimal.

Plus exactement, le temps de diffusion intra-grappes, appelé T_k, correspond aux prédictions des modèles de communication. Ce temps est obtenu grâce aux modèles présentés en chapitre 6. Cette notation T_k est équivalente à une notation où un noeud fictif k' est associé à chaque coordinateur k, dont :

$$L_{k,k'} + g_{k,k'} = \begin{cases} T_k & si\ k'\ est\ associe\ a\ k \\ \infty & pour\ tout\ autre\ processus\ j \neq k \end{cases}$$

Alors que les deux notations sont équivalentes, chacune a des avantages : l'adjonction d'un noeud fictif permet l'utilisation des heuristiques précédentes sans la nécessité d'une modification

des algorithmes, notamment le ECEF. L'utilisation de T_k permet une implantation plus simple des algorithmes, qui n'ont pas besoin de garder les deux identités *k* et *k'* associées à une grappe *k*. Si ces deux notations sont facilement transformées, nous avons gardé toutefois la description séparée *L*, *g* et *T*, afin de permettre une identification plus facile des facteurs évalués.

Dans ce sens, nous présentons deux nouvelles stratégies d'évaluation dites « sensibles au contexte des grilles de calcul », où le temps de diffusion *intra-grappe* est aussi considéré lors de la construction des arbres de diffusion. Pour mieux analyser l'efficacité de ces stratégies, nous avons aussi développé une version de l'heuristique ECEF-LA où le temps intra-grappes est pris en compte. Cette version sert de comparaison par rapport aux heuristiques de Bhat, présentées précédemment.

7.1.3.1 ECEF-LA*t*

L'heuristique ECEF-LA*t* est l'évolution naturelle de l'heuristique ECEF-LA où nous utilisons une fonction de *lookahead* adaptée à la représentation du coût de communication intra-grappes T_k.

Ainsi, l'heuristique ECEF-LA*t* cherche à minimiser le coût total de transmission et le temps nécessaire à la diffusion d'un message dans une grappe distante (en effet, le « petit *t* » du nom de cette heuristique indique qu'on cherche le minimum des temps). Pour cela, elle utilise une fonction d'évaluation :

$$F_j \quad = \quad \min_{P_k \in B} \left(g_{j,k}(m) + L_{j,k} + T_k \right)$$

À l'instar de l'heuristique ECEF-LA, le but de cette stratégie est que le récepteur soit choisi parmi les grappes qui peuvent retransmettre le message le plus vite possible à d'autres grappes. L'adjonction du temps T_k dans la fonction d'évaluation implique aussi que le choix d'un interlocuteur minimisera le temps de complétion des grappes contactées dans le futur.

7.1.3.2 ECEF-LA*T*

Une contrepartie de la technique précédente est que cette stratégie a tendance à favoriser les grappes rapides, ce qui peut entraîner des retards supplémentaires aux grappes plus lentes, reléguées aux dernières places. Pour éviter une telle situation, nous proposons une nouvelle fonction de *lookahead*, où le choix des grappes considère le maximum du temps nécessaire à la transmission et à la diffusion d'un message :

$$F_j \quad = \quad \max_{P_k \in B} \left(g_{j,k}(m) + L_{j,k} + T_k \right)$$

Malgré sa similarité avec l'heuristique précédente, cette nouvelle fonction d'évaluation cherche à équilibrer le temps de communication vers les différentes grappes, lentes ou rapides. En effet, nous cherchons dans un premier instant la grappe la plus lente qu'il reste à contacter (la fonction de *lookahead*), et parmi les choix d'émetteurs disponibles, nous choisissons celui qui peut la contacter le plus rapidement possible (fonction d'évaluation *min* de l'heuristique ECEF).

Le raisonnement de cette heuristique est que si les grappes les plus distantes ou les plus lentes (dans le sens où la diffusion des messages prend plus de temps) sont contactées en dernière place, leur diffusion prendra encore plus de retard, ce qui augmentera le temps d'exécution du Broadcast. Avec la fonction de *lookahead* de ECEF-LA*T*, nous choisissons comme récepteur la grappe qui prendra le moins de temps possible pour contacter la grappe la plus lente : cela garantit que si besoin est, les grappes les plus lentes seront contactées dans le minimum de temps possible.

7.1.3.3 BottomUp

La troisième heuristique proposée dans ce travail utilise une logique d'optimisation différente de celle utilisée par Bhat. En effet, l'approche de Bhat vise toujours la minimisation des facteurs liés à la transmission des messages et à sa diffusion, ce qui généralement finit par donner priorité aux grappes les plus rapides. Cependant, nous considérons que le temps d'exécution d'un Broadcast hiérarchique dépend surtout des grappes les plus lentes.

À partir des heuristiques précédentes, nous observons que, malgré l'utilisation de différentes fonctions de *lookahead*, les heuristiques de type ECEF-LA suivent toujours l'approche *min-max* ou *min-min*. Or, l'heuristique ECEF-LA*T* considère que parfois il est plus intéressant d'envoyer les messages d'abord aux grappes les plus lentes, pour ne pas retarder encore plus leur diffusion. D'autre part, il est aussi vrai qu'un grand nombre d'émetteurs favorise la conclusion rapide du Broadcast, et que l'envoi à des grappes plus lentes n'aide guère à augmenter le nombre d'émetteurs. Si ces deux raisonnements sont a priori opposés, ils ne sont pas incompatibles. En effet, les deux approches peuvent être combinées si des règles précises sont déterminées.

C'est ainsi que dans l'heuristique BottomUp nous définissons initialement une approche de type *max-min*, où l'émetteur est choisi parmi les grappes qui pourront contacter le plus rapidement possible la grappe la plus lente du réseau :

$$\max_{P_j \in B} \left(\min_{P_i \in A} \left(g_{i,j}(m) + L_{i,j} + T_j \right) \right)$$

En effet, cette approche permet que les grappes les plus lentes soient contactées dans le plus petit temps possible, ce qui peut minimiser le retard imputé à ces grappes. Toutefois, cette technique n'offre aucune garantie sur l'efficacité future des grappes émettrices. Pour cela, il serait peut-être intéressant d'ajouter une fonction de *lookahead*, à l'exemple des heuristiques de type ECEF-LA.

7.1.4 Simulations

Dans un premier moment, nous allons comparer ces différentes heuristiques à l'aide de simulations. À chaque itération, les paramètres L, g et T furent déterminés par un choix aléatoire uniforme sur des intervalles préalablement définis. Ces intervalles correspondent à des observations pratiques réalisées sur différents réseaux, et sont représentés dans Tableau 7.2.

TAB. 7.2 *Intervalles utilisés pour le tirage aléatoire des paramètres*

	minimum	maximum
L	0,001s (IDPOT <-> icluster2)	0,015s (IDPOT <-> GdX)
g	0,10s (IDPOT <-> icluster2)	0,60s (GdX <-> Rennes Parasoleil)
T	0,02s (broadcast 1Mo réseau Myrinet)	3s (broadcast 1Mo réseau Fast Ethernet)

Ces paramètres correspondent à des valeurs obtenues pendant l'expérimentation sur les différentes grappes de la grille Grid5000. Par exemple, les valeurs pour la latence et le gap ont été obtenues à partir d'expériences entre la grappe IDPOT et les grappes icluster2 et GdX. Les grappes IDPOT et icluster2 sont situées à Grenoble, alors que la grappe GdX est située à Orsay. La seule distance entre les différents sites donne déjà des latences très différentes (sans compter les caractéristiques des réseaux d'interconnexion qui différent également).

De la même manière, les valeurs pour T ont été obtenues à partir des mesures et prédictions réalisées dans le chapitre 6.

Pour évaluer le comportement des différentes heuristiques, nous avons simulé l'opération MPI_Bcast avec un nombre de grappes variant entre 2 et 50. Les valeurs présentées correspondent à la moyenne de 10000 simulations.

Dans un premier moment, notre analyse considérait un nombre relativement réduit de grappes, entre 2 et 10, car c'est le cas de la majorité des environnements actuels de type grille. Par exemple, la grille GRID5000 [GRI05] compte actuellement 10 grappes, et projette d'interconnecter 12 à 15 grappes courant 2006.

Ainsi, pour un nombre limité de grappes, comme illustré dans la Figure 7.1, les temps de complétion obtenus à partir de l'application des différentes heuristiques varient très peu. Cette différence est d'autant plus réduite du fait que les simulations ne peuvent pas tenir compte de certains aspects comme la complexité des heuristiques, qui peut avoir une influence sur le temps d'exécution de l'opération MPI_Bcast. Dans ce cas, où le nombre de grappes dépasse difficilement la dizaine, le facteur le plus important au moment du choix d'une heuristique est le rapport entre le coût de cette heuristique et le gain de performance qu'elle peut offrir, compte tenu de l'augmentation de la complexité de l'implantation de l'opération MPI_Bcast.

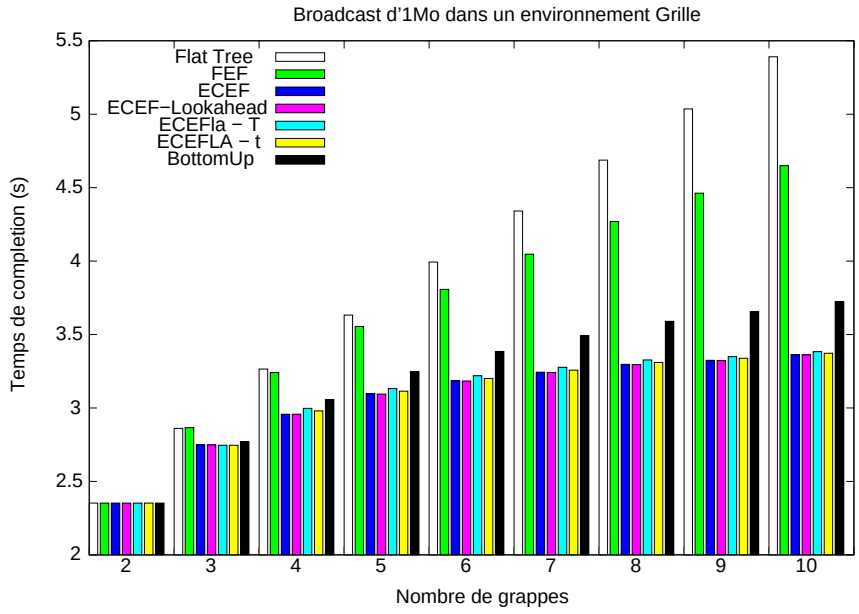


Figure 7.1 *Simulation des heuristiques pour le Broadcast avec 10 grappes*

D'un autre coté, notre analyse ne peut pas tenir compte seulement d'un nombre relativement réduit de grappes car, si pour l'instant les moyens informatiques disponibles ne sont pas au rendez-vous, il est probable que dans un futur proche le nombre de grappes (ou de grappes virtuelles) interconnectées sera bien plus élevé. C'est ainsi que nous montrons dans la Figure 7.2 les résultats des simulations de l'opération de Broadcast avec 50 grappes interconnectées.

Dans ce cas-là, nous observons plus clairement l'efficacité des différentes heuristiques de communication. Ainsi, l'analyse des valeurs présentées dans la Figure 7.2 démontre d'abord que l'heuristique Flat est clairement inefficace pour un grand nombre de grappes, car dans le cas général elle n'est pas suffisamment flexible pour profiter du recouvrement des communications entre les différentes grappes.

De la même manière, nous observons que l'algorithme glouton FEF n'est pas très efficace. Le choix de l'ordonnancement des communications sur le seul critère de la latence ne semble pas être

suffisant afin d'équilibrer les temps des différentes grappes, et par conséquent, le temps total d'exécution reste encore très dépendant du nombre de grappes interconnectées.

Dans un niveau intermédiaire, nous trouvons l'heuristique BottomUp. Non seulement elle est plus efficace que l'heuristique FEF, mais elle aussi semble plus capable de recouvrir les communications des différentes grappes. Cela démontre que la prise en charge des grappes les plus lentes dans le contexte de l'optimisation des communications est généralement un facteur aussi important que la vitesse de connexion entre les grappes. Un désavantage de cette approche toutefois est qu'elle ne garantit pas que les grappes contactées seront à leur tour des émetteurs rapides. Pour résoudre ce problème, une fonction de *lookahead* comme celle de l'heuristique ECEF-LA pourrait être utilisée.

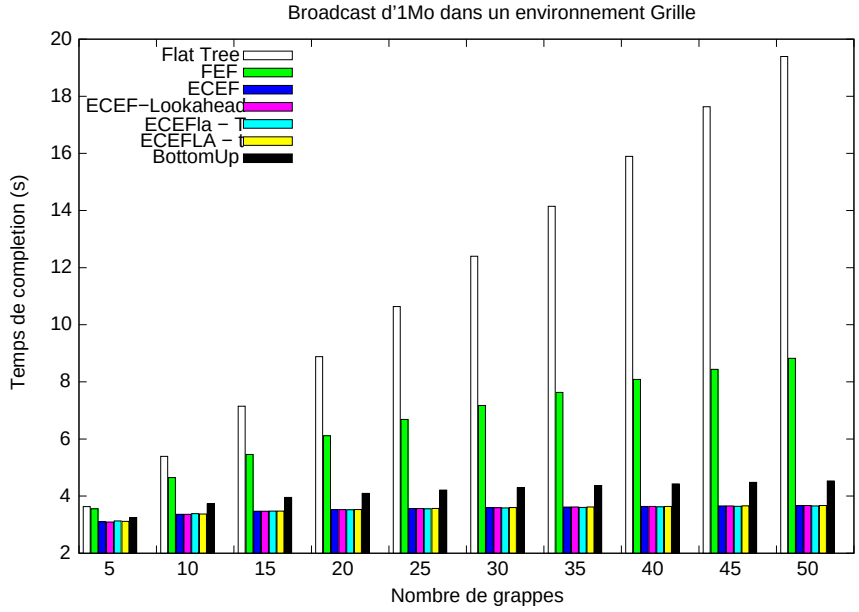


Figure 7.2 *Simulation des heuristiques pour le Broadcast avec 50 grappes*

Pour mieux analyser le comportement des heuristiques de type ECEF, la Figure 7.3 présente les résultats des simulations pour ces heuristiques.

Initialement, on peut constater que ces heuristiques utilisent largement le recouvrement de communication entre différentes grappes : l'augmentation du nombre de grappes n'est pas suivie d'une augmentation proportionnelle du temps de complétion de l'opération. En effet, la présence d'un nombre plus grand de grappes offre à ces heuristiques un choix beaucoup plus important de communications entre les grappes, ce qui favorise l'exécution de communications en parallèle.

On observe également que ces heuristiques présentent des performances moyennes très similaires. Cela est notamment vrai dans le cas des heuristiques ECEF-LA et ECEF-LAT, dont la différence repose sur la prise en charge du paramètre T_k. Cela indique que l'ordonnancement des communications des heuristiques évaluées est très similaire, malgré les différentes stratégies d'évaluation.

Comme les différentes techniques ont des performances moyennes très rapprochées, nous avons évalué ces heuristiques selon ce que nous avons appelé le « taux de réussite ». En effet, comme la détermination de l'ordonnancement optimal est une tâche très complexe, nous avons opté pour comparer les heuristiques par rapport au « minimum global ». Ce minimum correspond à la plus petite estimation obtenue avec les différentes heuristiques, à chaque itération. Le temps minimum est alors comparé avec les prédictions des différentes stratégies : c'est ainsi que la Figure 7.4 indique le nombre de fois que chaque stratégie atteint le minimum global, parmi les 10000 itérations effectuées.

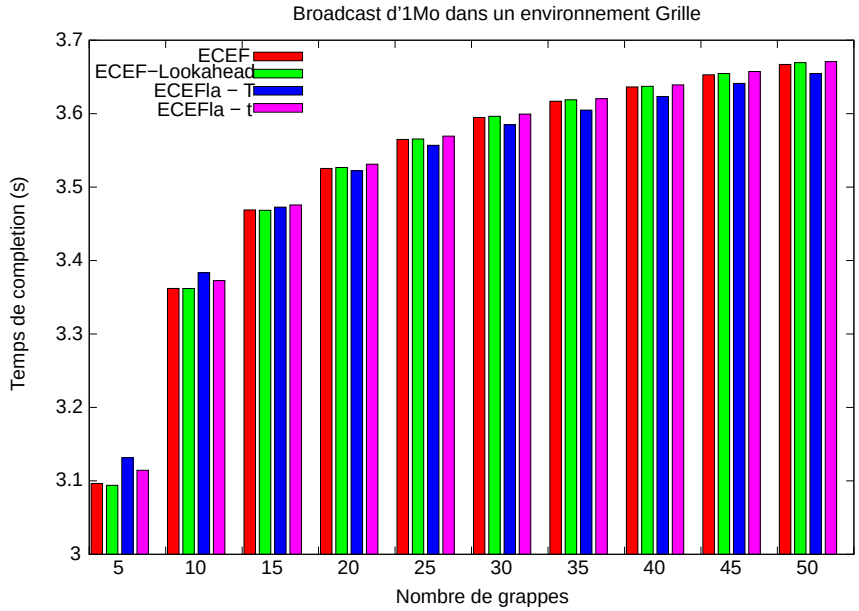


Figure 7.3 *Détail des simulations : Heuristiques de type ECEF*

À partir de l'analyse de la Figure 7.4 on peut observer que les heuristiques ECEF, ECEF-LA et ECEF-LAt ont des taux de réussite qui décroissent avec l'augmentation du nombre de grappes analysées. Cela est dû surtout à l'augmentation du nombre de choix d'ordonnancements possibles, qui renforce la tendance à choisir la valeur optimale locale, typique de ces heuristiques. C'est ce choix qui peut donner des ordonnancements avec des temps d'exécution légèrement écartés du minimal (sans pour autant influencer excessivement sur la moyenne).

En effet, l'analyse du taux de réussite indique qu'une seule heuristique garde un comportement constant, l'heuristique ECEF-LAT. Au contraire des autres techniques, cette heuristique maintient un taux de réussite autour de 45%. C'est à cause de cette stabilité que l'heuristique ECEF-LAT a une moyenne légèrement plus réduite que les autres quand le nombre de grappes est élevé, comme indiqué dans la Figure 7.3.

Ces résultats indiquent que le taux de réussite est un très fort indice de l'efficacité des heuristiques. Ainsi, afin d'avoir le maximum d'ordonnancements efficaces, l'utilisation mixte de différentes heuristiques selon le nombre de grappes est préconisée : des heuristiques simples quand le nombre de grappes est très réduit (trois ou quatre), les heuristiques ECEF ou ECEF-LA quand le nombre de grappes est peu élevé (cinq à vingt), et l'heuristique ECEF-LAT quand le nombre de grappes dépasse la vingtaine. Avec une telle combinaison des différentes heuristiques, nous pouvons toujours compter sur des ordonnancements efficaces.

7.1.5 Évaluation pratique

Si les simulations démontrent l'efficacité de certaines heuristiques, cela ne prouve pas qu'elles sont réellement efficaces en une situation pratique. En effet, l'application des heuristiques requiert une augmentation de la complexité des algorithmes de communication et du temps de calcul pour déterminer les ordonnancements des communications. Pour mieux évaluer ces heuristiques, nous avons étudié leurs performances sous trois configurations de grilles distinctes. Ces grilles utilisent les machines de plusieurs grappes liées au projet GRID5000 (IDPOT - Grenoble, *icluster2* - Grenoble,

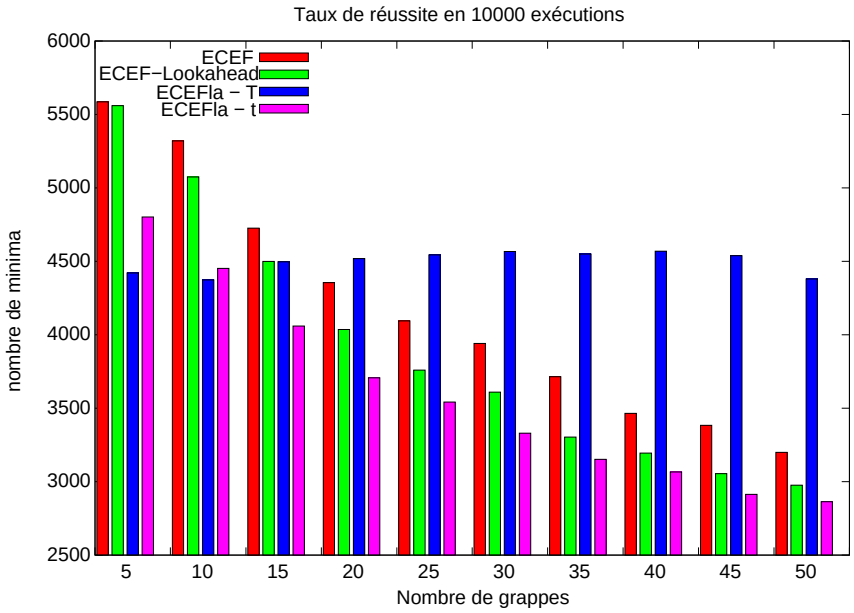


Figure 7.4 *Taux de réussite des heuristiques de type ECEF*

GdX - Orsay, Sophia-Antipolis et Toulouse). L'Annexe B contient un aperçu des valeurs mesurées du gap entre ces différentes grappes.

Les différentes heuristiques ont été implantées sur une version modifiée de la bibliothèque MagPIe que nous avons adapté pour l'acquisition et la manipulation des paramètres de communication entre les grappes. Pour cela, nous utilisons la procédure de découverte de topologie, présentée dans le chapitre 3. Cette procédure de découverte de topologie permet non seulement le regroupement des noeuds en grappes logiques homogènes (plus adaptées à la modélisation de performance), mais aussi fait automatiquement l'acquisition des paramètres pLogP correspondant à chaque sous-réseau homogène. Ces paramètres pLogP, une fois chargés en mémoire, sont associés à la structure hiérarchique du réseau décrite par la bibliothèque MagPIe. Ils sont utilisés pour prédire la performance des communications et pour choisir les meilleures stratégies selon les caractéristiques des réseaux.

Cas No. 1

La première expérience utilise 88 machines réparties entre les sites d'Orsay, Toulouse et Grenoble (IDPOT). Nous avons utilisé 60 machines de la grappe GdX d'Orsay, 20 machines de la grappe de Toulouse et 8 machines de la grappe IDPOT. À l'aide de notre outil de découverte de topologie *subnets*, ces machines ont été regroupées en 6 grappes homogènes différentes. La disposition des grappes, représentée dans la Figure 7.5, est fournie à la bibliothèque MagPIe à travers le fichier d'entrée *magpie_clusters*, et utilisé pour établir la répartition des processus sur différentes grappes.

Pour obtenir cette répartition des machines, on a utilisé l'algorithme de Lowekamp avec une tolérance de 30%. Alors que le Tableau 7.3 indique la latence entre deux sites différents ou entre deux machines d'un même site (à part les grappes qui ont une seule machine), nous constatons des variations importantes de performance pour le réseau IDPOT. En effet, la latence d'interconnexion entre les machines IDPOT varie entre de 35 μs et 242 μs, selon les machines. Ces variations sont dues

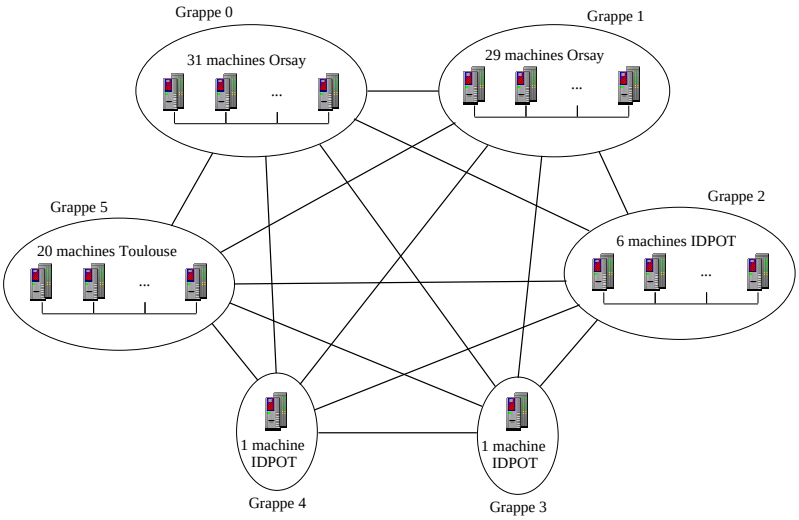


Figure 7.5 *Grille de 88 machines utilisée dans les expérimentations pratiques*

TAB. 7.3 *Latence entre les différents sites (en microsecondes)*

	Grappe 0	Grappe 1	Grappe 2	Grappe 3	Grappe 4	Grappe 5
	31 x Orsay	29 x Orsay	6 x IDPOT	1 x IDPOT	1 x IDPOT	20 x Toulouse
Grappe 0	47.56	62.10	12181.52	12187.24	12197.49	5210.99
Grappe 1	62.10	47.92	12181.52	12198.03	12195.22	5211.47
Grappe 2	12181.52	12181.52	35.52	60.08	60.08	5388.49
Grappe 3	12187.24	12198.03	60.08	0*	242.47	5393.98
Grappe 4	12197.49	12195.22	60.08	242.47	0*	5394.10
Grappe 5	5210.99	5211.47	5388.49	5393.98	5394.10	27.53

* cette grappe contient une seule machine.

surtout à l'utilisation de deux types différents de carte réseau, qui ont des performances distinctes, conformément à ce que nous avons indiqué dans [BAR04b].

Les mesures ont été effectuées en utilisant un processus de la grappe 0 comme racine, et leur résultat est présenté dans la Figure 7.6. On a comparé ces résultats avec l'implantation « pure » MPI, qui utilise des arbres binomiaux, comme décrit dans le chapitre 6. Les prédictions de performance ont utilisé les valeurs de *gap* et *latence* (conformément au modèle pLogP) mesurées lors de l'initialisation de l'application, à travers la procédure de découverte de topologie décrite dans le chapitre 3. Cependant, afin de permettre une meilleure compréhension de l'interconnexion entre les grappes, nous présentons les mesures des paramètres pLogP entre chaque grappe dans l'Annexe B.

Le premier résultat à noter est la faible performance de la stratégie Flat, même par rapport à l'implantation par défaut de LAM-MPI. Cela ne veut pas dire que la stratégie Flat est toujours moins performante que les autres stratégies, mais indique que cette stratégie est trop dépendante de la configuration du réseau, de l'ordre de représentation des grappes et du processus racine.

Dans le cas des autres heuristiques, on observe des gains de performance déjà très importants. L'heuristique BottomUp, comme prévu par les simulations, n'est pas aussi efficace que les autres heuristiques, qui de leur côté, se comportent de manière très similaire.

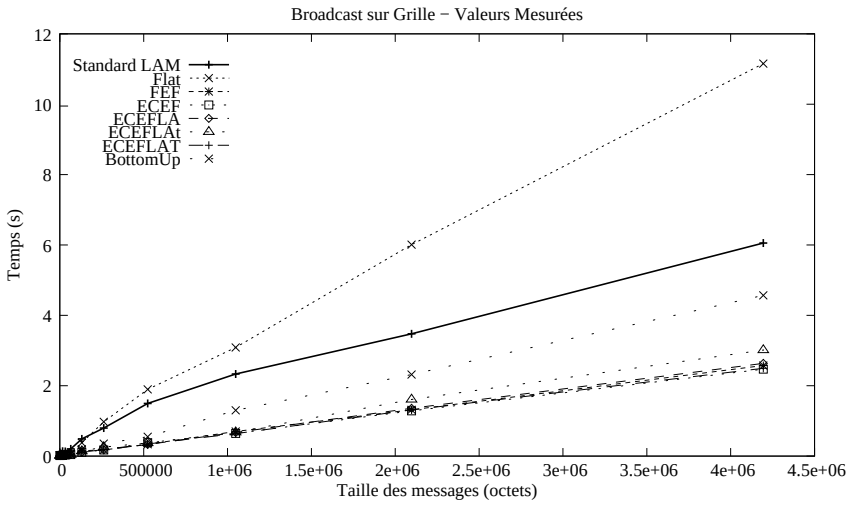


Figure 7.6 *Performance du Broadcast sur une grille de 88 machines*

Le faible écart observé entre les prédictions des heuristiques de type FEF et ECEF-* est justifié surtout par le nombre réduit de grappes, qui réduit le nombre de combinaisons possibles et fait converger les résultats des différentes heuristiques.

Pour mieux valider les résultats des expériences, la Figure 7.7 présente les temps prévus des différentes heuristiques. Ces temps, calculés automatiquement par les heuristiques d'ordonnancement des communications, donnent une meilleure indication de la fiabilité des modèles par rapport aux résultats pratiques. Dans ce cas, nous observons que les heuristiques de type FEF et ECEF-* ont des résultats très rapprochés, certainement parce qu'elles ont obtenu le même ordonnancement des communications. D'un autre côté, l'écart entre ces prédictions et les résultats réels sont bien plus importants pour les heuristiques FEF et ECEF-* que pour le BottomUp ou le Flat. Cela indique que le coût du calcul de l'ordonnancement et le coût de la mise en oeuvre de ces communications sont les facteurs les plus importants, et reflètent l'augmentation de complexité d'une communication à couches multiples.

Cas No. 2

La deuxième expérience utilise 78 machines réparties entre les sites d'Orsay, Toulouse, Sophia-Antipolis et Grenoble (IDPOT). Alors que dans le cas précédent la majorité des machines appartenait à une seule grappe (Orsay), cette fois-ci nous avons utilisé des grappes plus équilibrées. En effet, nous avons utilisé 20 machines de la grappe GdX d'Orsay, 20 machines de la grappe de Toulouse, 19 machines de la grappe de Sophia-Antipolis et 17 machines de la grappe IDPOT.

Comme dans le cas précédent, ces machines ont été regroupées en 6 grappes homogènes différentes à l'aide de notre outil de découverte de topologie, qui implante l'algorithme de *clustering* de Lowekamp. La disposition des grappes, représentée dans la Figure 7.8, est utilisée par la bibliothèque MagPIe pour établir la répartition des processus sur différentes grappes.

Les latences d'interconnexion entre les différents sites sont présentées dans le Tableau 7.4. On observe que malgré la tolérance de 30% utilisée par l'algorithme de *clustering*, la différence entre les machines IDPOT est suffisamment élevée pour forcer le regroupement des machines en plusieurs grappes.

Le résultat des mesures effectuées est présenté dans la Figure 7.9. On a comparé ces résultats avec l'implantation « pure » MPI, qui utilise des arbres binomiaux. Cette fois-ci, la disposition des

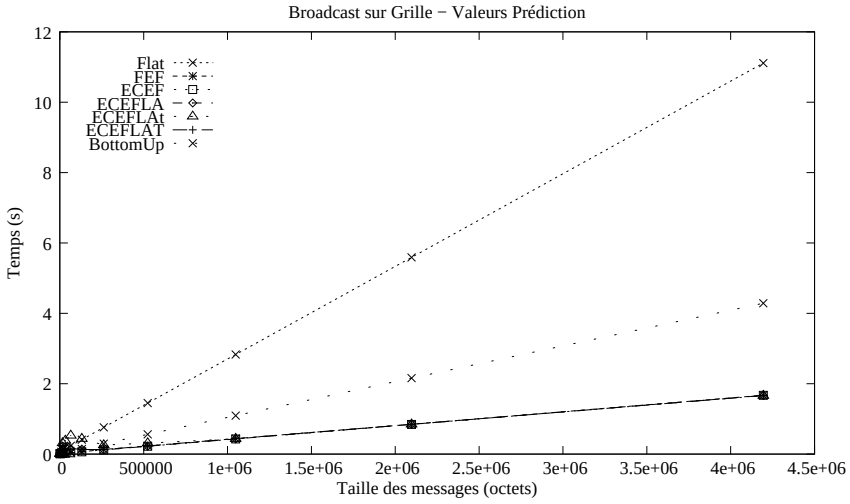


Figure 7.7 *Prédictions pour une grille avec 88 machines*

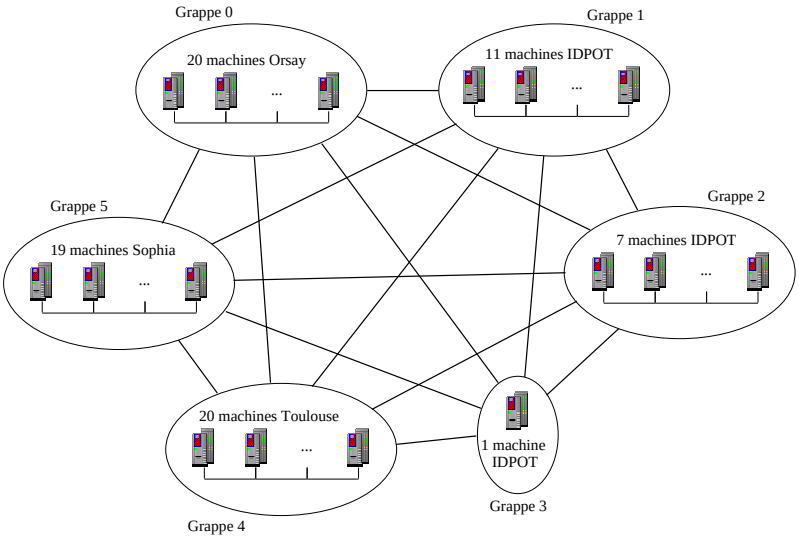


Figure 7.8 *Grille de 78 machines*

grappes dans le fichier d'entrée de MagPIe a favorisé la stratégie Flat, qui a obtenu des performances similaires à celle des heuristiques plus élaborées.

Dans un premier temps, l'analyse de la Figure 7.10 indique que les heuristiques permettent un gain de performances important par rapport à l'algorithme en arbre binomial de la bibliothèque LAM-MPI. Parmi ces heuristiques, nous observons que les stratégies qui considèrent seulement la vitesse des liens, à l'instar de l'heuristique FEF, n'atteignent pas la meilleure performance. En effet, celle-ci est atteinte par les heuristiques de type ECEF, qui considèrent la disponibilité des processus, et non seulement la vitesse des liens.

Une remarque importante est que dans ce cas la stratégie Flat a obtenu des résultats similaires à ceux des meilleures heuristiques. Cela ne veut pas dire que l'heuristique Flat est optimale, mais

TAB. 7.4 *Latence entre les différents sites (en microsecondes)*

	Grappe 0	Grappe 1	Grappe 2	Grappe 3	Grappe 4	Grappe 5
	20 x Orsay	11 x IDPOT	7 x IDPOT	1 x IDPOT	20 x Toulouse	19 x Sophia
Grappe 0	48.39	6577.49	6586.49	6592.51	5211.94	8602.73
Grappe 1	6577.49	35.52	59.96	59.96	5387.48	2736.56
Grappe 2	6586.49	59.96	60.08	79.51	5393.98	2740.26
Grappe 3	6592.51	59.96	79.51	0*	5405.78	2745.98
Grappe 4	5211.94	5387.48	5393.98	5405.78	26.94	3630.51
Grappe 5	8602.73	2736.56	2740.26	2745.98	3630.51	35.04

* cette grappe contient une seule machine.

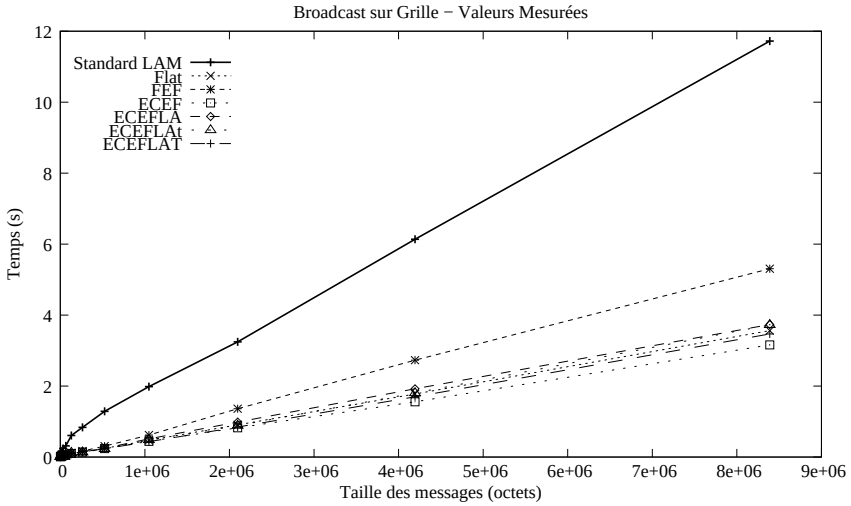


Figure 7.9 *Performance du Broadcast sur une grille de 78 machines*

seulement qu'elle est adaptée à l'ordre des grappes et au processus racine utilisé dans cette expérience.

Cas No. 3

La dernière expérience utilise 190 machines réparties entre les sites d'Orsay, Toulouse, Sophia-Antipolis et Grenoble (*icluster2*). Nous avons choisi ce dernier cas d'étude afin d'évaluer le comportement des heuristiques par rapport à un grand nombre de machines (le double des expériences précédentes). Pour cela, la grappe GdX a fournie 50 machines, la grappe de Toulouse avec 28 machines, la grappe de Sophia-Antipolis avec 51 machines et finalement la grappe *icluster2* avec 60 machines.

Les machines ont été regroupées en 5 grappes homogènes, comme représentée dans la Figure 7.11, et cette description de la topologie est utilisée par la bibliothèque MagPIe pour établir la répartition des processus sur différentes grappes.

Le résultat des mesures effectuées est présenté dans la Figure 7.12. Comme dans le cas précédent,

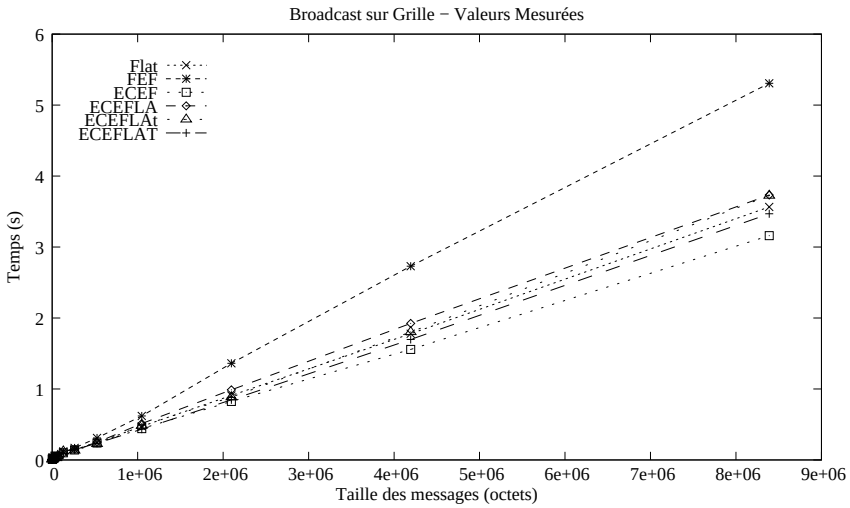


Figure 7.10 *Détails des performances des heuristiques pour le Broadcast*

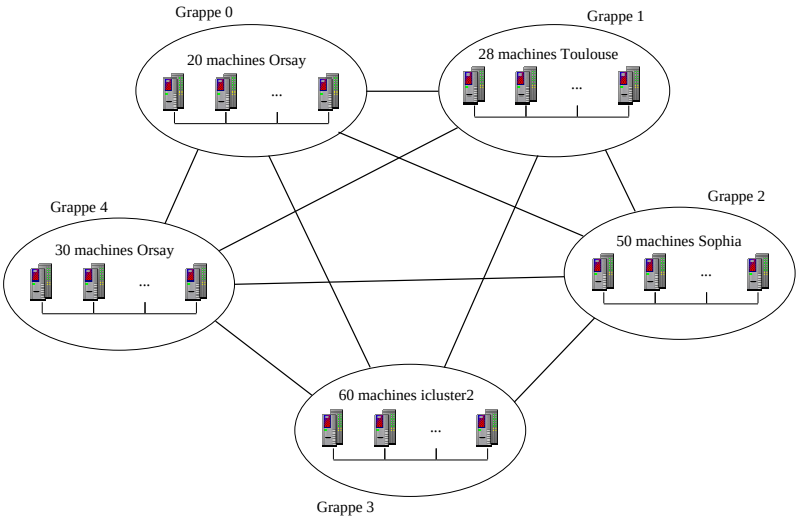


Figure 7.11 *Grille de 190 machines*

le temps nécessaire à l'opération par défaut de LAM-MPI est bien plus élevé que celui obtenu avec des heuristiques d'optimisation pour les grilles.

L'analyse de la Figure 7.13 indique que cette fois-ci la représentation des grappes et le choix du processus racine favorise les heuristiques Flat et FEF. Cela ne démérite pas les autres heuristiques, qui ont obtenu également des bonnes performances.

Parmi ces heuristiques, le BottomUp et le ECEFLA-T ont obtenu des performances assez intéressantes. En effet, dans cet environnement, les grappes sont composées d'un nombre important de noeuds, ce qui représente un temps de diffusion plus important à l'intérieur des grappes. Or, les heuristiques BottomUp et ECEFLA-T sont justement les stratégies où le coût des grappes les plus lentes est mieux pris en compte, et cela justifie les bons résultats obtenus par ces heuristiques.

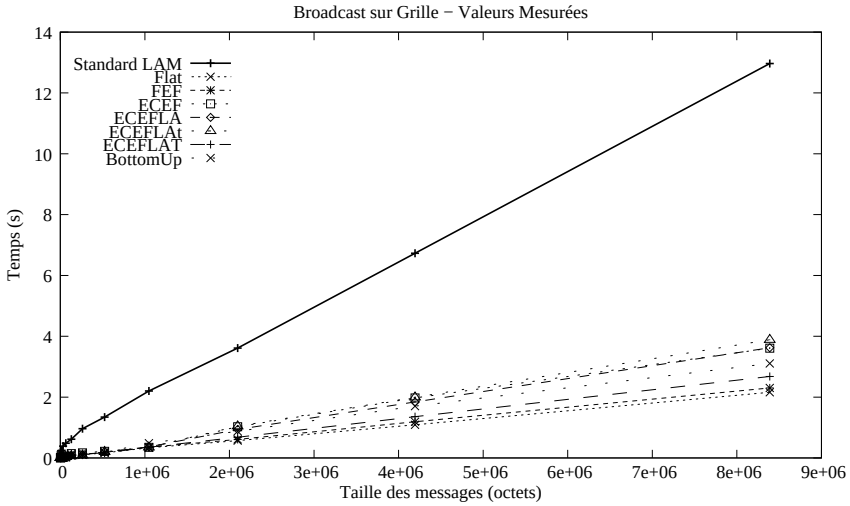


Figure 7.12 *Performance du Broadcast sur une grille de 190 machines*

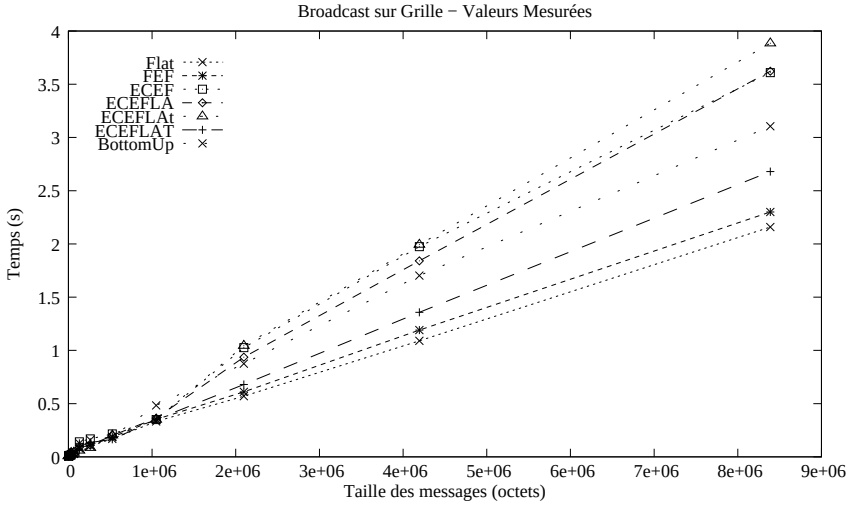


Figure 7.13 *Détails des performances des heuristiques pour le Broadcast*

7.1.6 Considérations sur l'opération MPI_Bcast

Si dans un premier temps l'analyse des trois cas d'étude permet la validation des certaines prédictions faites à travers la simulation de différents environnements réseau, son apport le plus important est l'observation du comportement des implantations des heuristiques.

Les trois situations étudiées ont notamment mis en évidence l'importance du processus racine et de la répartition des processus sur des différentes grappes sur la performance des stratégies plus simples. En effet, la performance de la stratégie Flat est fortement liée à l'ordre de représentation des grappes répartition, généralement fournie par l'utilisateur. De surcroît, la stratégie Flat utilise toujours le même ordre de diffusion, indépendamment du rang du processus racine. Au contraire, les heuristiques les plus élaborées construisent des arbres de diffusion adaptés à chaque situation, ce

qui rend possible un gain de performance plus important, surtout quand le rôle de *racine* est alterné entre plusieurs processus.

D'ailleurs, les simulations indiquent que la performance des stratégies simples, comme le Flat, supportent très mal l'augmentation du nombre de grappes interconnectées. Ceci met en cause l'efficacité de ces stratégies dans un futur proche. Ainsi, nous croyons que, même si la grille compte un nombre réduit de grappes, l'utilisation d'une heuristique un peu plus élaborée, comme par exemple l'heuristique ECEFLA-T, offre le meilleur rapport coût-bénéfice-robustesse.

7.2 Scatter

7.2.1 État de l'art

Comparé aux autres opérations de communication collective comme le Broadcast, le Gather, le Reduce ou le All-to-All, le Scatter fait rarement partie des sujets de recherches sur l'optimisation des communications. Cela est dû, en partie, au manque relatif de flexibilité de l'algorithme, qui doit envoyer des messages différents à chaque processus, et qui pour cette raison ne peut pas utiliser toutes les stratégies employées avec le Broadcast. D'un autre côté, les communications engendrées par le patron de communication *Un vers Plusieurs Personnalisé*, représenté par l'opération Scatter, ne sont pas suffisantes pour surcharger le réseau, au contraire du patron *Plusieurs vers Plusieurs Personnalisé*, qui est souvent à l'origine d'importants effets de bord à cause de la congestion du réseau.

Dans le cadre des réseaux hétérogènes, notamment les grilles de calcul, peu de travaux ont étudié des optimisations pour l'opération Scatter. Parmi ces travaux, nous identifions deux approches différentes. La première approche, inspirée par les implantations classiques du Scatter, vise l'établissement de connexions directes entre le processus *racine* et les destinataires des messages. Dans ce cas-là, les optimisations visent notamment la minimisation du nombre de retransmissions subies par un message. La deuxième approche est inspirée à la fois par la structure hiérarchique du réseau et par les techniques utilisées par le Broadcast. Dans ce cas, l'objectif est la multiplication des sources de transmission et la meilleure utilisation du débit des réseaux.

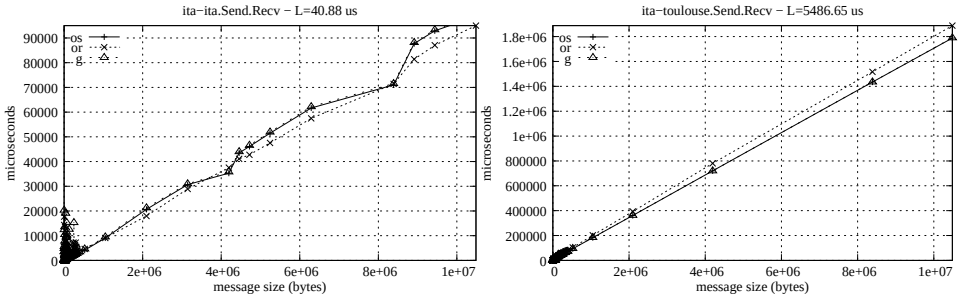
Dans les sections suivantes nous présentons quelques techniques qui illustrent ces deux approches.

7.2.1.1 Communication directe

L'approche utilisée par la bibliothèque MagPIe version 2.0.2 est similaire à l'approche par défaut de LAM-MPI ou MPICH, où le processus *racine* établit une communication directe vers les autres processus. Le raisonnement de cette approche est que si chaque message est destiné à un seul processus, le moyen le plus rapide de lui envoyer le message est d'établir un contact direct.

Cependant, l'établissement de plusieurs connexions induit une surcharge importante sur le processus *racine*, surcoût qui risque de devenir encore plus grand avec la taille d'une grille de calcul. D'autre part, l'établissement de plusieurs connexions n'est valable que si le coût de transfert des messages est proportionnel à sa taille. En réalité, le protocole TCP/IP requiert une période de temps important pour atteindre son efficacité maximale sur des liens de longue distance. À cause des techniques de contrôle de congestion, les connexions TCP augmentent progressivement le débit (dans la phase appelée *slow start*) selon le taux d'acquittement des paquet. Il faut alors normalement compter six à huit allers-retours sur le réseau avant d'atteindre le régime « stationnaire ». Pour cette raison, l'ouverture d'une connexion TCP pour l'envoi d'un petit message est normalement bien plus chère que si le message était plus grand.

Finalement, cette stratégie ne tient pas compte des caractéristiques du réseau, ce qui ne permet pas un ordonnancement efficace des transmissions.



(a) Intra-grappe (icluster2) (b) inter-grappes (icluster2->toulouse)

Figure 7.14 *Paramètres pLogP pour des connexions intra et inter-grappes*

7.2.1.2 Segmentation des messages

Comme observé précédemment, les travaux que visent l'optimisation de l'opération Scatter dans les environnements homogènes sont peu nombreux, et cette constatation est aussi valable pour les environnements hétérogènes.

Un des rares travaux qui discutent l'optimisation du Scatter pour les grilles de calcul est celui de Kielmann *et al.* [KIE01]. Dans le travail de Kielmann, l'optimisation consiste à segmenter les messages avant leur transmission, qui s'effectue selon la stratégie en arbre plat. Le raisonnement de cette approche est que la segmentation des messages permettrait une meilleure utilisation du débit du réseau local. Cela est possible grâce à un processus *passerelle* responsable du routage des messages vers les réseaux distants, tel un commutateur de type *store and forward*. Ce routage libère alors l'émetteur aussi tôt que la *passerelle* reçoit le message.

Bien que la segmentation des messages dépende de certains facteurs liés aux caractéristiques du réseau, elle s'avérer intéressante. Par exemple, une caractéristique qui peut influencer la performance est l'existence de l'effet *passerelle*, c'est à dire, l'émetteur est libéré aussi tôt que la *passerelle* reçoit le message. Les mesures que nous avons effectué indiquent que le processus émetteur reste « bloqué » pendant la transmission du message vers un réseau distant. La Figure 7.14 présente un exemple de ces mesures, où le coût de communication vers un réseau distant est plus important que celui à l'intérieur du réseau local.

Ainsi, si la segmentation des messages est une alternative intéressante dans certains cas, son utilisation dépend des caractéristiques du réseau. D'autre part, elle requiert une implantation plus complexe, et compte tenu de certaines restrictions de temps, les expériences exécutées dans le cadre de ce travail ont porté sur d'autres techniques d'optimisation plus classiques.

7.2.1.3 Communication hiérarchique

À partir de l'hypothèse que l'envoi de grands messages est généralement plus avantageux que l'envoi de plusieurs petits messages, l'utilisation d'une diffusion hiérarchique semble envisageable, à l'exemple de l'opération Broadcast. Dans ce cas, les messages destinés à une grappe sont d'abord regroupés, avant d'être envoyés comme un seul message, comme déjà suggéré par Miled [MIL98].

Cependant, nous ne pouvons pas oublier que le problème du Scatter est la généralisation du problème de diffusion. Par conséquent, la recherche d'un patron de diffusion optimal pour le Scatter est plus complexe que celle du Broadcast, qui est déjà NP-Complet. De ce fait, la recherche d'une hiérarchie de communication à plusieurs niveaux pour le Scatter requiert une analyse approfondie des facteurs qui peuvent influencer le temps d'exécution du Scatter. D'un côté, la retransmission des messages représente un facteur prépondérant dans le coût de l'opération Scatter, et la structuration

des communications en plusieurs niveaux ne fait qu'augmenter le nombre de retransmissions. De l'autre côté, la construction d'une hiérarchie à plusieurs niveaux exige que le coût des transmissions soit recalculé à chaque fois qu'une grappe est rajoutée à une branche (Figure 7.15).

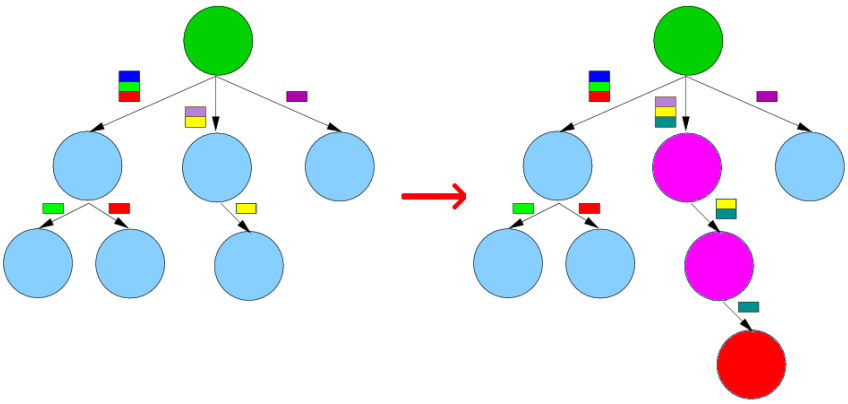


Figure 7.15 *Impact d'une solution à multi-couches sur l'algorithme du Scatter*

Comme cette analyse demande un effort considérable, nous n'avons pas pu élaborer des stratégies efficaces dans le temps de cette thèse. Cependant, des algorithmes simples à deux couches de communication sont facilement envisageables. Nous avons alors procédé à l'évaluation d'un algorithme en deux couches où les messages destinés à une grappe spécifique sont regroupés et envoyés au coordinateur de cette grappe. À son tour, chaque coordinateur fait la diffusion des messages à l'intérieur des grappes en utilisant la stratégie la plus adaptée à son réseau (l'arbre plat ou l'arbre binomial), identifiée à partir des données d'interconnexion.

7.2.1.4 Ordonnancement des communications

Si l'utilisation d'une hiérarchie de communication à deux niveaux permet déjà l'obtention d'un gain de performance grâce au regroupement des messages, elle sous-utilise les connaissances du système dont nous disposons, notamment l'étendue de l'opération à l'intérieur d'une grappe. À partir de ces informations, nous avons utilisé une heuristique gloutonne, fondée sur les mêmes principes de l'heuristique *Slowest Node First* proposée par Liu [LIU00a] dans le cadre de l'opération Gather. Dans l'heuristique qu'on a appelée *Grappe Plus Lente d'Abord* (référence à *Slowest Cluster First*), les informations d'interconnexion sont utilisées pour déterminer non seulement le temps d'exécution du Scatter à l'intérieur des grappes, mais aussi le temps nécessaire à la transmission d'un paquet qui contient tous les messages destinés à ces grappes. Cela dit, l'heuristique détermine d'abord le temps minimum nécessaire à chaque grappe pour finir la diffusion des messages, son *ready-time*.

À partir de ces données, les grappes sont ordonnées en ordre décroissant de *ready-time*, de manière à ce que les grappes les plus lentes soient contactées en priorité. Le principe utilisé est que tout retard survenu avant la communication avec les grappes les plus lentes repousse encore plus leur terminaison. Or, le simple fait de contacter d'autres grappes au lieu des plus lentes entraîne déjà des retards (correspondant au *gap* des communications).

Cette technique a quelques désavantages. La transmission en bloc de tous les messages destinés à une grappe peut représenter un temps important, qui n'est que légèrement compensé par la meilleure performance de la communication de grands messages. Une manière de réduire ces temps de communication et d'augmenter le recouvrement des communications serait l'envoi de « segments » de blocs, suffisamment grands pour bénéficier du débit de transmission mais encore petits pour favoriser le « *pipeline* ». Cependant, cette alternative requiert une coordination bien plus importante au niveau de l'algorithme de Scatter, et pour cette raison nous ne l'avons pas implantée et évaluée.

TAB. 7.5 *Latence entre les différents sites (en microsecondes)*

	Grappe 0	Grappe 1	Grappe 2	Grappe 3	Grappe 4	Grappe 5
	20 x Orsay	11 x IDPOT	7 x IDPOT	1 x IDPOT	20 x Toulouse	19 x Sophia
Grappe 0	48.39	6577.49	6586.49	6592.51	5211.94	8602.73
Grappe 1	6577.49	35.52	59.96	59.96	5387.48	2736.56
Grappe 2	6586.49	59.96	60.08	79.51	5393.98	2740.26
Grappe 3	6592.51	59.96	79.51	0*	5405.78	2745.98
Grappe 4	5211.94	5387.48	5393.98	5405.78	26.94	3630.51
Grappe 5	8602.73	2736.56	2740.26	2745.98	3630.51	35.04

* cette grappe contient une seule machine.

7.2.2 Comparaison des stratégies

Premier cas

La première expérience utilise 78 machines réparties entre les sites d'Orsay, Toulouse, Sophia-Antipolis et Grenoble (IDPOT). Pour cela, la grappe GdX d'Orsay a fournie 20 machines, la grappe de Toulouse avec 20 machines, la grappe de Sophia-Antipolis avec 19 machines et finalement la grappe IDPOT avec 17 machines.

À l'aide de notre outil de découverte de topologie *subnets*, présenté dans le chapitre 3, ces machines ont été regroupées en 6 grappes homogènes. Cette répartition des grappes (cf. Figure 7.16) est utilisée par la bibliothèque MagPIe pour établir la répartition des processus sur différentes grappes, dont les latences d'interconnexion entre les différents sites sont présentées dans le Tableau 7.5 (les autres paramètres d'interconnexion, dont le gap, sont présentés dans l'Annexe B).

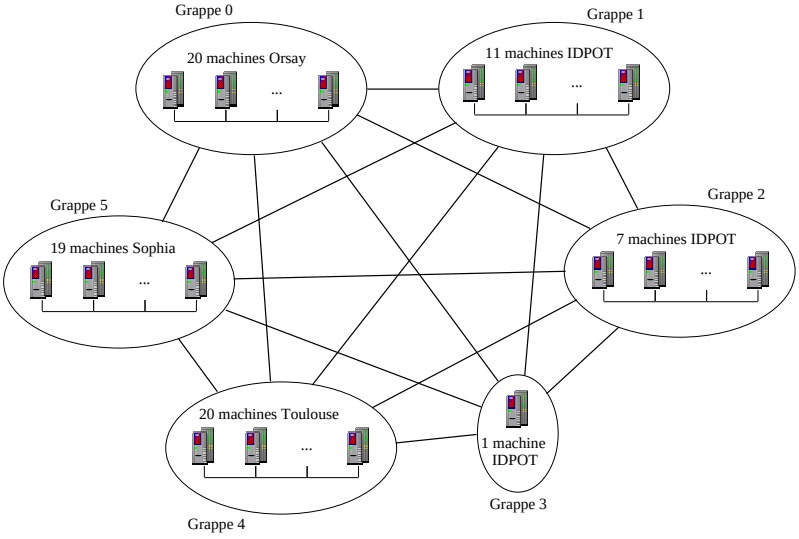


Figure 7.16 *Grille de 78 machines*

Le résultat des mesures effectuées est présenté dans la Figure 7.17. Le gain de performance des stratégies hiérarchiques est bien évident, ce qui démontre que les techniques de regroupement des

messages permettent une meilleure utilisation du débit des liens de longue distance.Toutefois, la transmission groupée des messages a un coût, qui est notamment important dans le cas des messages plus petits que 100 Ko : dans ce cas-là, le temps nécessaire au regroupement des messages et à sa transmission dépasse le coût de la transmission directe.

Pour finir, nous avons observé que les expériences n'ont pas indiqué un vrai avantage pour la stratégie *Slowest Cluster First*, comme prévu. Toutefois, le nombre de grappes qui composent la grille est réduit, et des gains de performance pourraient être possibles avec un nombre de grappes plus important.

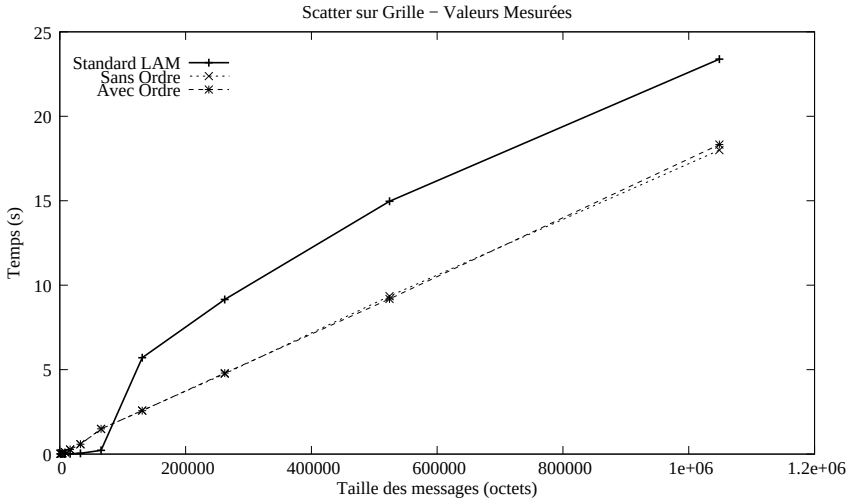


Figure 7.17 *Performance du Scatter sur une grille de 78 machines*

Deuxième cas

La dernière expérience utilise 208 machines réparties entre les sites d'Orsay, Toulouse, Sophia-Antipolis et Grenoble (IDPOT et *icluster2*). Pour cela, la grappe GdX d'Orsay a fournie 49 machines, la grappe de Toulouse avec 29 machines, la grappe de Sophia-Antipolis avec 52 machines et finalement les grappes *icluster2* et IDPOT avec 60 et 16 machines respectivement.

Les machines ont été regroupées en 6 grappes homogènes, conformément à la Figure 7.18. Une curiosité de cet environnement est qu'une partie des machines de la grappe IDPOT a été regroupée avec les machines de la grappe *icluster2*, évidence qui prouve que la localisation des machines n'est pas un paramètre suffisant pour établir des grappes homogènes.

Le résultat des mesures effectuées est présenté dans la Figure 7.19. Cette fois-ci encore, le temps nécessaire à l'opération par défaut de LAM-MPI est bien plus élevé que celui obtenu avec des heuristiques d'optimisation pour les grilles, si la taille des messages dépasse les 100 Ko.

Ce surcoût de l'opération de LAM-MPI peut avoir plusieurs causes. Parmi les possibilités, le remplissage de la fenêtre de transmission TCP semble être l'un des facteurs le plus importants. En effet, le débit maximal atteint est une fonction directe du taux d'acquittement des messages, qui est limité par la latence de transmission des messages. Dans le cas de l'implantation Scatter de LAM-MPI, l'ouverture de plusieurs connexions limite encore plus la capacité de réagir à ces acquittements, alors que dans le cas de la transmission groupée, la connexion dure assez longtemps pour qu'un enchaînement des transmissions soit établi.

Autre détail important, le temps nécessaire à la terminaison de l'opération MPI_Scatter avec nos heuristiques est pratiquement le même dans les deux cas étudiés, malgré la variation sensible du

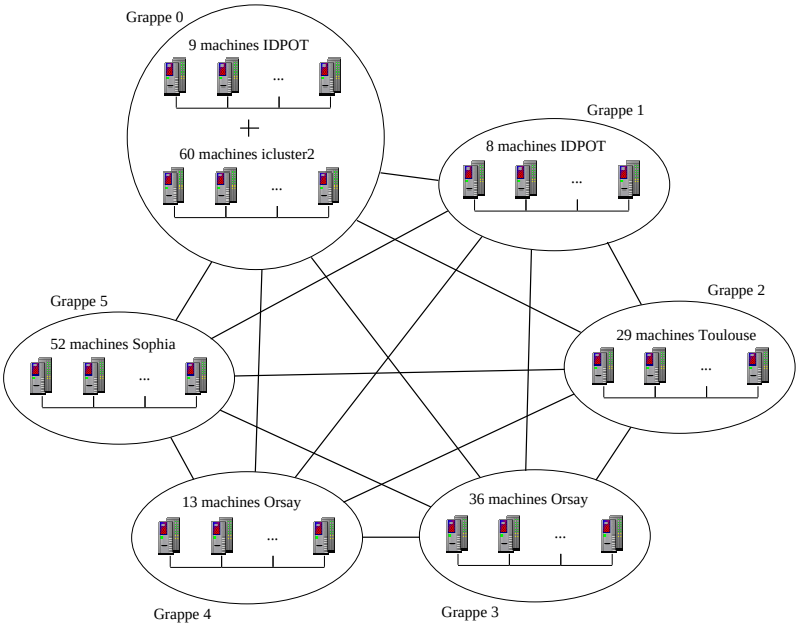


Figure 7.18 *Grille de 208 machines*

nombre de machines. Comme dans le cas précédent, le regroupement préalable des messages permet une meilleure utilisation du débit des liens de longue distance, réduisant le temps nécessaire à la terminaison de l'opération MPI_Scatter.

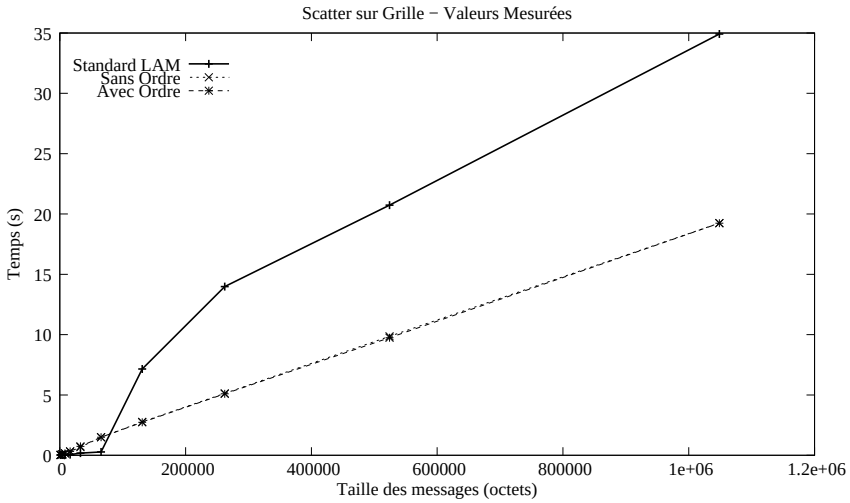


Figure 7.19 *Performance du Scatter sur une grille de 208 machines*

7.2.3 Considérations à propos de l'opération MPI_Scatter

L'analyse des résultats indique que l'efficacité de l'opération MPI_Scatter dans le contexte d'un environnement de type grille de calcul réside dans la capacité d'envoi du processus racine. L'agrégation des messages permet des gains de performance importants, tandis que l'ordonnancement des communications permet surtout l'amélioration de ces performances. Malheureusement, ces performances restent encore trop dépendantes du débit maximal atteint par le processus racine, surtout parce que l'opération MPI_Scatter ne se prête pas à une organisation hiérarchique à multiples couches de communication, comme l'opération MPI_Bcast.

À partir de ces observations, on a envisagé une manière de contourner ces limitations, à travers la multiplication des sources de transmission. Cette technique, inspirée de l'opération de broadcast, devient efficace si le coût de transmission à l'intérieur d'une grappe est vraiment réduit par rapport au coût de transmission vers d'autres réseaux distants.

Dans ce cas, une partie des données est partagée avec un ou plusieurs processus du même réseau du processus *racine*, qui feront à leur tour la redistribution vers les autres grappes. L'avantage de cette technique est l'utilisation plus efficace du lien VTHD avec la multiplication des sources de transmission, car en général le débit maximal d'un seul noeud n'atteind pas la capacité des liens d'interconnexion entre les grappes. De cette manière, le nombre de transmissions simultanées est augmenté ainsi que le débit de transfert. Tout cela se fait pour un surcoût relativement réduit (la transmission vers des processus du réseau local).

Cependant, l'implantation de cette technique reste assez complexe : pour bien mener l'opération, les sources doivent se coordonner pour définir les destinataires que chacune doit contacter, et les récepteurs doivent avoir une connaissance *a priori* du rang de leur processus émetteur. D'autre part, le nombre de sources doit être choisi de manière à ce que la technique reste efficace malgré le coût de distribution des données vers les sources secondaires. Heureusement, cette dernière tâche peut être facilement modélisée, à l'instar du choix de la meilleure taille de segment pour le broadcast segmenté (cf. page 92).

7.3 All-to-All

Parmi les opérations de communication collective, le All-to-All est à la fois l'une des opérations les plus générales et aussi l'une des plus complexes à modéliser. En effet, tous les processus s'engagent dans un processus d'échange de données où les messages différent selon le destinataire.

Dans le chapitre 6 on a montré que la modélisation de l'opération MPI_Alltoall ne peut pas se faire de manière traditionnelle, car les communications sont fortement influencées par la congestion du réseau. Ce constat a orienté certains auteurs à développer des stratégies de communication pour les réseaux commutés, où les communications sont ordonnées de manière à ne pas surcharger un lien (ou commutateur) identifié comme un goulot d'étranglement.

Dans le cas des environnements de type grille, le facteur le plus important n'est plus la congestion du réseau (malgré sa présence), mais la variation des performances de communication entre les différentes grappes. En effet, à cause de l'hétérogénéité du réseau, les processus qui engagent un échange pair-à-pair peuvent être sujets à des communications déséquilibrées, où l'un des processus est plus lent que l'autre.

Une solution intéressante à ce problème est l'équilibrage des échanges de données. Dans ce cas, des techniques d'équivalence (« *matching* » en anglais) sont alors utilisées pour minimiser le déséquilibre des communications et maximiser l'utilisation du temps.

7.3.1 Équivalence « pair-à-pair »

L'implantation typique de l'opération MPI_Alltoall utilise un modèle d'échange « pair-à-pair » (*pairwise exchange* en anglais). Plusieurs bibliothèques MPI, comme par exemple MPICH, LAM-

MPI et même la bibliothèque pour grilles MagPIe utilisent ce concept, avec quelques petites variations. Dans ces bibliothèques, les interlocuteurs et l'ordre utilisé pour leur communication sont déterminés de manière statique, ayant comme seul paramètre le rang des différents processus.

Si cette méthode est suffisamment adéquate pour un environnement homogène, elle peut s'avérer trop contraignante dans le cas des systèmes hétérogènes. En effet, le principe de l'échange « pair-à-pair » est que le temps de communication entre les différents interlocuteurs est similaire et, par conséquent, une étape de communication entre $\frac{P}{2}$ paires de processus peut se faire dans un intervalle de temps T. À la fin d'une étape de communication, tous les processus concernés sont prêts à engager la communication avec d'autres interlocuteurs.

Dans un environnement hétérogène, par contre, le temps nécessaire à l'échange de données entre deux paires de processus différentes n'est plus le même, du fait que les temps de communication sont différents. À cause de cette variation des temps de communication, les étapes de communication ne sont plus synchronisées, comme le montre la Figure 7.20, et souvent les interlocuteurs doivent attendre que leurs paires terminent l'étape précédente de communication.

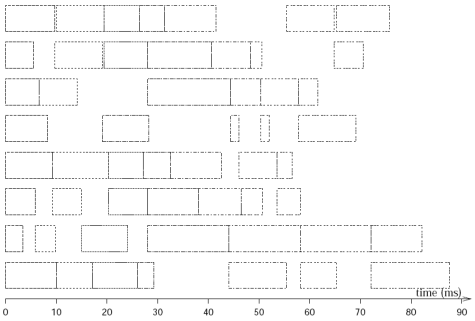


Figure 7.20 *Exemple d'ordonnancement avec la stratégie fixe [GOL05]*

Pour minimiser le temps d'attente entre les différentes étapes de communication, une solution possible est l'harmonisation des communications de manière qu'à chaque étape de communication, le temps nécessaire aux différentes paires de processus soit le plus proche possible. Seulement, étant donné que le problème de l'Échange Total est NP-complet [BHA98], l'utilisation d'heuristiques d'optimisation s'impose.

Dans ce sens, Bhat [BHA98] a proposé trois heuristiques. La première, *maximum matching*, est une heuristique fondée sur l'algorithme de maximisation du poids (*maximum weight*) qui a une complexité de $O(P^4)$. Cet algorithme utilise un graphe biparti où l'arc entre le noeud v_i à gauche et le noeud v_j à droite a un poids équivalent au temps de communication entre les processus P_i et P_j. Ceci dit, une étape de communication valide contient P arcs qui correspondent à la permutation de $(P_0, ..., P_{P-1})$ processus. Pour trouver cette combinaison, Bhat utilise un algorithme de *maximum weight complete matching*, qui a une complexité de $O(P^3)$. L'algorithme de Bhat consiste alors en P exécutions de l'algorithme *maximum weight* où, à chaque itération, les arcs correspondants à l'étape de communication obtenue sont supprimés du graphe biparti. Cet ordonnancement de communications n'impose aucune synchronisation entre les phases, car celles-ci ne sont initialisées dès lors que l'émetteur et le récepteur sont prêts.

Le deuxième algorithme est une approximation gloutonne de l'algorithme *maximum weight*, qui s'exécute en $O(P^3)$ itérations. Initialement, les événements de communication de chaque processus sont triés par ordre décroissant du temps de communication. Pour construire une étape de communication, l'algorithme traverse la liste d'événements de chaque processus à la recherche d'un processus destinataire. Un processus destinataire est alors le premier processus dans la liste que n'avait pas encore été sélectionné, et qui n'est pas encore le destinataire d'un autre processus. Par contre, si à la

fin de la recherche il n'a pas été possible de définir un destinataire, ce processus reste inactif pendant cette étape de communication. À cause de ces étapes de communication creuses, le nombre total d'étapes de communication peut être plus grand que P. Cependant, pour garantir l'équité, un processus qui était inactif pendant une étape de communication sera le premier à choisir un destinataire dans le tour suivant. Si aucun processus était inactif, alors le dernier processus d'une étape sera le premier processus de l'étape suivante.

Finalement, l'heuristique *Open Shop* est une technique fondée sur le problème d'optimisation *Open Shop*, très similaire au problème d'ordonnancement des communications du All-to-All. Cette heuristique a une complexité de $O(P^3)$. Dans cet algorithme, chaque processus i garde une liste R_i avec les processus qu'il doit contacter, ainsi que deux tableaux $sendavail[]$ et $recvavail[]$, où est indiquée la disponibilité de chaque émetteur et récepteur. Par exemple, *sendavail[i]* indique l'instant à partir duquel le processus i peut participer à une opération d'envoi (similaire à la variable *ReadyTime* utilisée avec l'heuristique de broadcast ECEF).

Pour ordonnancer les communications, cet algorithme recherche un processus émetteur i qui sera disponible à partir de l'instant *sendavail[i]*. Ensuite, sa liste de récepteurs R_i est parcourue pour trouver le récepteur j disponible le plus tôt possible. Ainsi, la communication entre i et j est programmée pour l'instant $t = max(sendavail[i], recvavail[j])$, et les valeurs de *sendavail[i]* et de *recvavail[j]* sont mises à jour avec la valeur de t, alors que j est supprimé de la liste R_i.

D'une manière similaire, Goldman *et al.* [GOL05] ont proposé certaines heuristiques afin de résoudre le problème du All-to-All, à la différence près qu'ils se préoccupent du problème de l'opération MPI_Alltoallv dans le contexte des environnements homogènes. Dans ce cas, les différentes tailles de message sont responsables par la variation des temps de communication entre les paires. Goldman a étudié trois heuristiques, *max-min*, *max-weight* et *uniform*.

La technique *max-weight* est similaire à celle de Bhat, et cherche à maximiser la somme des poids des communications. L'heuristique *max-min* a pour objectif la construction d'une séquence de communications où le poids minimum des communications est maximisé, comme l'illustre la Figure 7.21. En effet, la différence entre ces deux techniques est la manière dont le flux de données est considéré : *max-weight* cherche à envoyer le plus de données possible dans une étape de communication, maximisant le débit effectif de chaque phase. La technique *max-min*, par contre, cherche à équilibrer les temps de communications de tous les processus, minimisant l'écart entre les différents processus et réduisant ainsi le temps en attente. Ces deux techniques ont une complexité de $O(P^3)$.

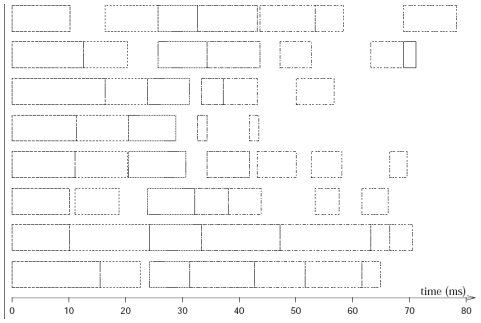


Figure 7.21 *Exemple d'ordonnancement avec la stratégie* max-min *[GOL05]*

La dernière technique étudiée par Goldman, appelée *uniform*, cherche une séquence de communications où le temps de toutes les communications est le même. Pour cela, cet algorithme fait l'usage de la segmentation des messages qui sont alors découpés selon les besoins de l'algorithme et peuvent être envoyés en plusieurs étapes de communication non nécessairement contiguës. Cette procédure, similaire à celle étudiée par Liu *et al.* [LIW96], a pour grand avantage la minimisation

du temps d'attente des processus. Cependant, la segmentation des messages a tendance à produire des segments trop petits (cf. Figure 7.22), dont le coût de transmission est proportionnellement plus élevé que celui des messages plus grands. Pour cette raison, Goldman *et al*. proposent l'utilisation mixte de la technique *uniform* et d'une autre technique comme le *max-weight*, plus efficace pour des petits messages.

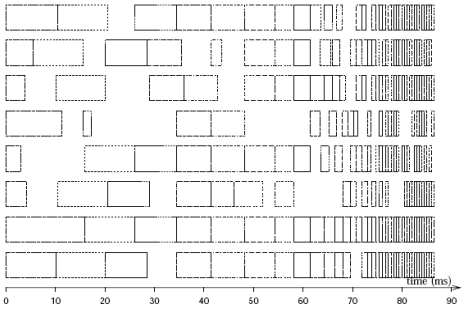


Figure 7.22 *Exemple d'ordonnancement avec la stratégie* uniform *[GOL05]*

Heureusement, les problèmes de Bhat et Goldman sont équivalents. En effet, leur objectif est la réduction du temps d'attente entre les différentes étapes de communication. Dans le cas de Bhat, le temps de communication entre les processus est représenté directement dans la matrice de distances utilisée par les heuristiques, alors que dans le cas de Goldman, ce temps est représenté indirectement par la taille des messages (qui sont le seul responsable pour la variation du temps de communication dans un réseau homogène). Il suffit alors de transformer la taille des messages en temps de communication pour avoir les mêmes paramètres du problème de Bhat. Plus important, cette équivalence regroupe les opérations All-to-All et All-to-Allv sous la même classe de problèmes, et par conséquent, des solutions pour l'une des opérations peuvent être utilisées aussi pour l'autre opération.

7.3.2 Évaluation pratique

Pour cette expérience, on utilise 40 machines réparties également entre les sites de Toulouse, Sophia-Antipolis et Grenoble (IDPOT). Les latences d'interconnexion entre les différents sites, présentées dans le Tableau 7.6, ont été obtenues par l'outil de détection de topologie *subnets*, qui implante l'algorithme de *clustering* de Lowekamp [LOW00]. Cet algorithme, utilisé avec une valeur de ρ égal à 30% a permis la séparation des noeuds en 3 grappes homogènes, comme illustré dans la Figure 7.23.

TAB. 7.6 *Latence entre les différents sites (en microsecondes)*

	Grappe 0	Grappe 1	Grappe 2
	13 x IDPOT	7 x IDPOT	20 x Sophia
Grappe 0	35.52	59.96	2735.97
Grappe 1	59.96	60.08	2744.07
Grappe 2	2735.97	2744.07	34.92

Dans un premier moment, nous avons comparé la performance des deux implantations typiques du MPI_Alltoall présentes dans les bibliothèques LAM-MPI et MPICH. Cette approche a été choisie car nous ne disposons pas d'un modèle de communication qui permet une prédiction de performance

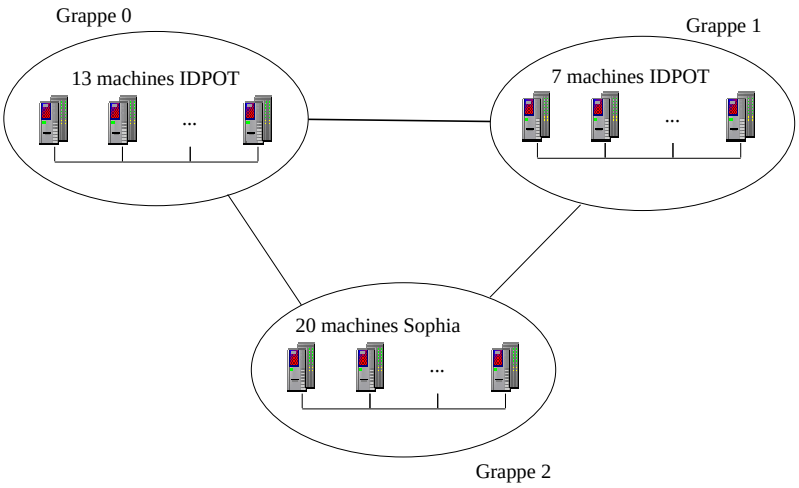


Figure 7.23 *Grille de 40 machines*

pour le All-to-All dans un environnement hétérogène. Ces deux implantations utilisent la communication directe entre les paires de processus, ayant comme seule différence l'ordre de contact : la méthode « Échange Direct Optimisé » tente d'éviter la surcharge d'un processus spécifique en alternant l'ordre des interlocuteurs selon le rang des processus.

À l'exemple de ce qui fut observé précédemment dans le contexte des grappes, on n'a pas descellé des différences suffisamment importantes entre les deux approches, comme indiquent les résultats présentés dans la Figure 7.24. Nous avons ensuite évalué les heuristiques d'équivalence *max-min* et *max-weight* proposées par Goldman [GOL05]. Puisque ces techniques ne font pas la segmentation des messages, leur implantation est beaucoup plus simple que l'heuristique *uniform*.

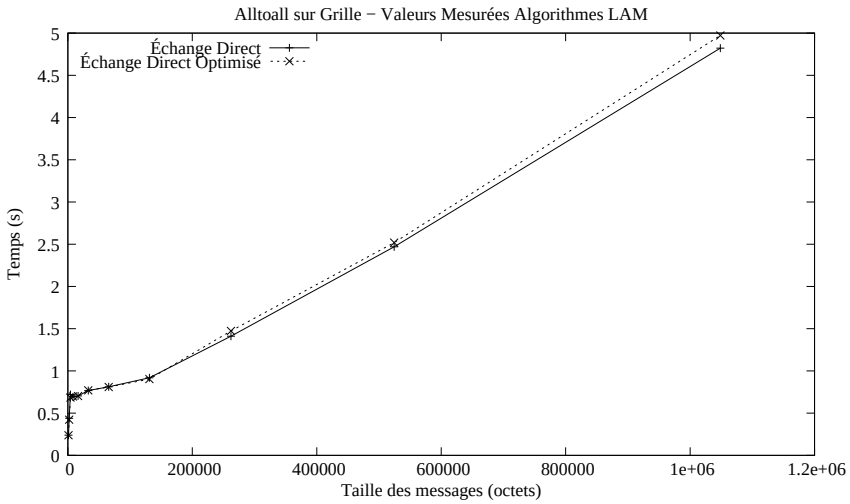


Figure 7.24 *Performance des algorithmes d'échange direct pour le All-to-All*

Encore une fois, nous n'avons pas pu desceller des différences significatives entre les deux heuristiques. Malgré un léger avantage pour l'heuristique *max-weight*, comme l'indique la Figure 7.25,

les temps obtenus avec les deux heuristiques sont trop proches pour indiquer quelle heuristique est la plus efficace.

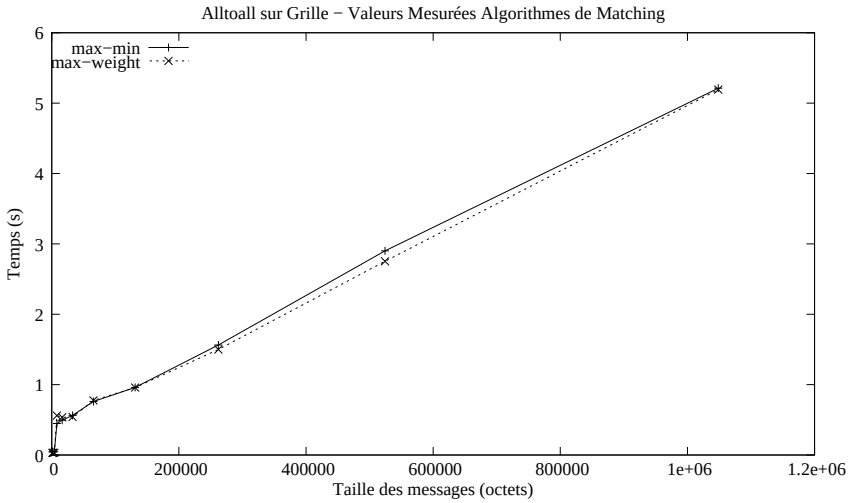


Figure 7.25 *Performance des algorithmes d'équivalence pour le All-to-All*

La performance de ces heuristiques, par rapport aux implantations par défaut de MPI semble mitigée : malgré l'avantage théorique apporté par l'ordonnancement des communications, leur performance est presque similaire à celle des algorithmes *Échange Direct*.

Dans une comparaison entre les différentes stratégies, présentée dans la Figure 7.26, nous observons que la performance des heuristiques d'équivalence est légèrement inférieure à celle des algorithmes d'échange direct pour des grands messages. Cependant, quand les messages échangés ont moins de 100 Ko, les algorithmes d'équivalence (max-min et max-weight) ont l'avantage (conforme Figure 7.27).

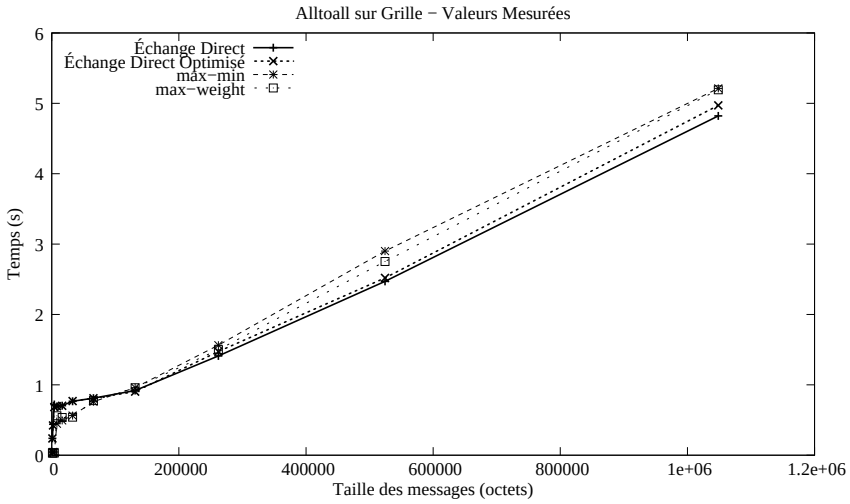


Figure 7.26 *Comparaison entre les différentes approches étudiées*

Une explication possible pour le comportement presque identique des différentes stratégies, optimisées ou non, est que malgré la distance entre les grappes, qui induit une hétérogénéité importante, la communication est fortement restreinte par la capacité d'envoi sur le réseau, ceci malgré les apports des heuristiques d'équivalence. En effet, nous observons que l'utilisation des algorithmes d'équivalence permet un certain gain de performance dans le cas des petits messages, où la congestion du réseau est moins probable.

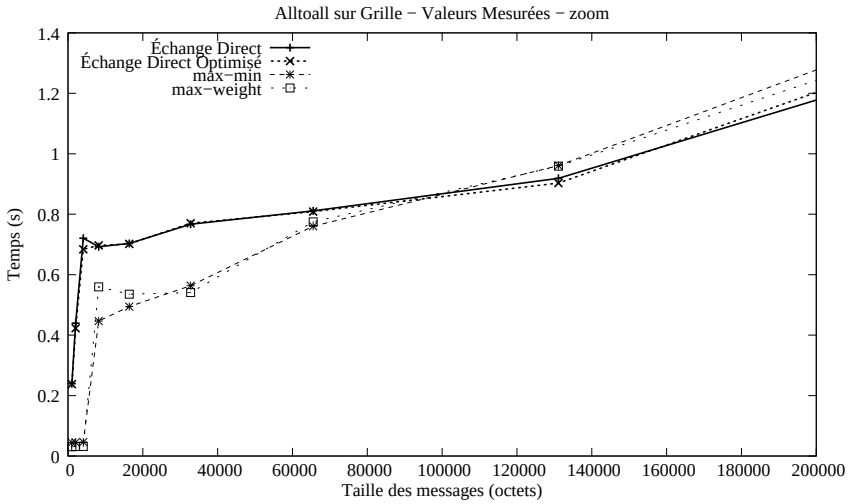


Figure 7.27 *Comparaison entre les différentes approches étudiées pour des petits messages*

C'est pour cette raison que dans nos travaux futurs nous aimerions évaluer des situations encore plus hétérogènes et propices aux algorithmes d'équivalence, comme par exemple l'opération MPI_Alltoallv. Nous espérons pouvoir établir plus exactement l'impact de l'hétérogénéité et de la congestion du réseau sur la performance des heuristiques d'optimisation.

7.3.3 Considérations à propos de l'opération MPI_Alltoall

L'opération de communication collective MPI_Alltoall représente le patron d'échange de messages le plus complexe dans un environnement distribué. Le succès des techniques d'optimisation ne dépend plus uniquement de la diffusion rapide des informations, mais également de la prise en charge des aspects de congestion du réseau et de surcharge des noeuds. Très souvent, ce patron de communication atteint la saturation des liens de communication, d'où la nécessité de bien ordonnancer les communications, surtout si l'environnement est hétérogène.

À partir de cette constatation, nous avons évalué des techniques d'ordonnancement réputées pour minimiser le déséquilibre des connexions. Les résultats obtenus sont assez révélateurs : ils indiquent que l'hétérogénéité du réseau n'est qu'un facteur à considérer, et que la congestion du réseau peut jouer un rôle très important sur l'efficacité des heuristiques d'optimisation. Il est donc très important de continuer la recherche dans ce domaine et de quantifier l'importance de chacun des facteurs, congestion et hétérogénéité.

7.4 Discussion

Dans ce chapitre, nous avons étudié l'impact de différentes techniques d'optimisation pour les communications collectives sur grilles de calcul. En effet, cette étude a analysé des stratégies qui

pourraient notamment tirer avantage des informations fournies par les modèles de performance étudiés auparavant.

Certains patrons de communication, notamment le Broadcast et le Scatter, ont pu bénéficier d'importants gains de performance par rapport aux techniques traditionnelles. D'autres opérations, comme le All-to-All, ont présenté des résultats moins évidents, indiquant que chaque patron de communication nécessite une attention individualisée.

Néanmoins, les expériences effectuées ont démontré que la modélisation des communications intra-grappes et inter-grappes est essentielle pour l'optimisation des opérations de communication collective sur grille. Nous avons pu démontrer, notamment dans le cas de l'opération MPI_Bcast, que ces informations de performance peuvent être utilisées pour composer des communications hiérarchiques particulièrement efficaces. La précision et le faible coût des modèles de communication permettent aux différentes heuristiques testées de bien exploiter la capacité des grilles de calcul.

Il faut aussi mettre en évidence le comportement des nouvelles heuristiques proposées dans ce chapitre. Ces heuristiques ont démontré, à travers des simulations et des tests pratiques, des performances équivalentes à celles des meilleures heuristiques existantes, ayant pour avantage une plus grande fiabilité par rapport au nombre de grappes interconnectées. Avec la constante augmentation des ressources de calcul mis en commun par les institutions scientifiques, comme par exemple la grille GRID5000, les techniques d'optimisation devront désormais être aussi efficaces pour un nombre réduit de grappes que pour tout un univers de ressources de calcul.

8

Conclusions et Perspectives

8.1 Travaux présentés

Les travaux présentés dans cette thèse ont porté sur la modélisation de performance des communications collectives et les difficultés soulevées par l'hétérogénéité intrinsèque aux plates-formes à grande échelle dans le cadre du calcul distribué et parallèle.

Afin d'optimiser la performance des opérations de communication collective dans une grille de calcul, la modélisation des performances est un outil essentiel. Toutefois, l'hétérogénéité de cet environnement rend le problème de l'optimisation des performances très dur. Pour simplifier le problème et permettre un gain de performance dans ces environnements, nous avons étudié dans cette thèse l'hypothèse des communications organisées en multiples couches hiérarchiques. Dans cette hypothèse, un réseau hétérogène peut être modélisé à partir des performances individuelles de chaque composant. Nous avons augmenté cette définition, de manière à ce que la performance d'une grille de calcul puisse être décrite par l'ordonnancement des communications des grappes qui font partie de cette grille.

C'est ainsi que la première partie de cette thèse présente une étude sur la découverte de topologie du réseau. Dans le chapitre 3, nous étudions le cas spécifique de notre thèse, où la découverte de topologie est utilisée pour le regroupement des machines d'une grille de calcul en grappes homogènes, procédure qui facilite la modélisation des performances. Nous avons constaté que certaines propriétés nécessaires à notre travail ne sont pas respectées par les outils traditionnels de découverte de topologie, et pour cela, nous avons développé notre propre méthodologie de découverte de topologie, qui mélange à la fois des techniques traditionnelles et des techniques de mesure de performance. Cette méthodologie (cf. [BAR04b]) est ainsi adaptée aux besoins de performance et précision des applications parallèles.

Dans l'Annexe A, en revanche, nous illustrons l'importance de la découverte de topologie dans un autre domaine, celui des systèmes distribués. Nous étudions notamment le problème de la Diffusion Totalement Ordonnée, un problème classique qui subit des importantes limitations causées par le passage à l'échelle (*scalability*) [BAR03]. Nous montrons comment la découverte de topologie peut être utilisée pour aider à minimiser le coût de cette opération dans le cadre d'un réseau géographiquement distribué.

Une fois que les aspects liés à l'hétérogénéité du réseau ont été traités, la deuxième partie de cette thèse s'est intéressée sur les aspects liés à la modélisation des communications collectives à

l'intérieur des grappes homogènes. Nous avons étudié trois importants patrons de la communication collective, *Un vers Plusieurs*, *Un vers Plusieurs Personnalisé* et *Plusieurs vers Plusieurs Personnalisé*. Pour cela, nous avons analysé plusieurs techniques couramment utilisées pour l'implantation de ces patrons de communication. Des modèles de performance correspondants à ces techniques ont été développés, et ensuite validés avec des expérimentations réelles.

Dans le cas des patrons *Un vers Plusieurs* et *Un vers Plusieurs Personnalisé*, notre apport s'est concentré sur la proposition de différentes stratégies de communication et leur validation. Ceci représente une relecture et une extension des modèles existants dans la littérature, et nous avons pu montrer que l'utilisation de modèles de communication est une alternative efficace pour identifier les stratégies de communication les plus adaptées aux caractéristiques de l'environnement parallèle utilisé. Ces résultats ont été présentés dans plusieurs publications, dont [BAR04a],[BAR04c] et [BAR05a].

Dans le cas du patron de communication *Plusieurs vers Plusieurs Personnalisé*, en revanche, nous avons observé que les modèles couramment cités dans la littérature ne sont pas capables de représenter les conditions réelles de congestion du réseau. À partir de ces observations, nous avons proposé un nouveau modèle de performance, mieux adapté aux situations réelles. Ce modèle de performance est une évolution des modèles traditionnels ([CHR99, PJE05]), et a subi une évolution graduelle, comme l'attestent nos publications sur le sujet, [BAR04c],[BAR05a] et [BAR05b].

Une fois avoir établi les modèles de performance pour les opérations collectives dans les environnements homogènes, nous nous sommes intéressés à l'optimisation des communications dans un environnement du type grille de calcul. Parmi les différentes techniques, nous avons étudié notamment les heuristiques d'ordonnancement du type hiérarchique. En effet, nous avons travaillé avec l'hypothèse que les communications collectives sur les grilles de calcul peuvent être organisées en plusieurs couches hiérarchiques. Cette hypothèse représente une évolution par rapport aux implantations classiques, où les communications sont divisées seulement en deux couches, *intra* et *inter*-grappes.

Finalement, la dernière partie de cette thèse a étudié l'impact d'une organisation hiérarchique des communications. Pour cela, nous avons utilisé des heuristiques pour l'ordonnancement des communications, adaptées à l'environnement des grilles de calcul. Comme dans la partie précédente, nous avons cherché à valider ces heuristiques à travers l'expérimentation sur un environnement réel, de manière à évaluer l'efficacité des optimisations par rapport aux prédictions des modèles de performance.

8.2 Perspectives

Les résultats de ces travaux ouvrent de nombreuses perspectives, dont quelques unes seront présentées dans cette section.

Dans un avenir proche, l'un des objectifs principaux est la mise en oeuvre d'autres expériences sur l'environnement des grilles de calcul, afin de mieux valider les techniques implantées. Pour cela, nous comptons faire des expériences avec des applications de calcul parallèle réelles, pour mieux analyser l'impact des optimisations sur le temps d'exécution des applications. D'autre part, nous considérons qu'il faudrait approfondir l'analyse d'efficacité des heuristiques du Broadcast lorsque plusieurs processus racine sont utilisés tout au long de l'exécution de l'application. Cette analyse est très importante pour bien souligner l'avantage de certaines techniques, qui s'adaptent aux situations, alors que d'autres (notamment la technique en arbre Plat) sont plus dépendantes de la topologie fournie.

Un autre sujet qui doit également être traité est l'optimisation de la découverte de topologie. Dans le modèle actuel, les mesures sont faites automatiquement au démarrage de l'application MPI, ce qui peut représenter un surcoût important selon la durée des applications. En augmentant les informations contenues dans le fichier de description de topologie, nous pourrions donner à l'utili-

sateur le choix entre un démarrage rapide et une mesure plus précise des paramètres, faite au début de l'exécution. Cela donne aussi la possibilité de comparer l'impact des différents modèles de coût (dont pLogP et Hockey) sur la modélisation des communications collectives.

À plus long terme, nous souhaiterions également approfondir la recherche sur les heuristiques d'optimisation et les modèles de performance. Dans le premier cas, nous nous sommes intéressés à l'évaluation de certaines techniques proposées dans le chapitre 7, dont l'utilisation de fonctions de *lookahead* avec l'heuristique BottomUp sur le Broadcast, ou la multiplication de sources dans le cas du Scatter. Par rapport à la modélisation des performances, nous aimerions étudier plus profondément la possibilité de modéliser l'opération All-to-All sur les grilles de calcul, à l'instar de ce qui a déjà été fait avec les opérations Broadcast et Scatter. À cet objectif s'ajoute l'extension des modèles de performance, afin d'étendre les heuristiques d'ordonnancement des communications pour le cas des machines multiprocesseurs, qui peuvent exécuter plusieurs processus simultanément.

Finalement, un autre axe complémentaire à nos expériences est l'implantation d'autres opérations de communication collective adaptées aux grilles de calcul. Certaines d'entre elles, comme le Reduce, le Gather, le Barrier et les opérations avec taille de message variable peuvent bénéficier directement des techniques étudiées dans cette thèse.

IV

Annexes

Annexe A

Topologie pour la Tolérance aux Fautes

La rapide expansion des systèmes comme les grilles de calcul a créé non seulement des problèmes de performance, mais aussi de sûreté de fonctionnement des applications. Les techniques de tolérance aux fautes, largement utilisées dans le développement des systèmes répartis, ont malheureusement une utilisation très restreinte dans le cadre des environnements à large échelle, notamment à cause du coût des techniques traditionnelles et de la volatilité des ressources dans ces systèmes [MAR04]. Certaines implantations de MPI comme MPICH-V [BOS02], FT-MPI [FAG03] ou OpenMPI [GAB04] permettent l'utilisation de techniques de tolérance aux fautes, notamment le *checkpointing* (sauvegarde) [ELN96][SCH02]. Malgré ces efforts, la plupart de ces techniques sont encore très coûteuses, et certaines applications nécessitent d'autres garanties que celles offertes par le *checkpointing*.

Un exemple de service de tolérance aux fautes encore très peu répandu sur les environnements de grille de calcul est l'opération de Diffusion Totalement Ordonnée (aussi connue comme Diffusion Atomique), requise par la plupart des systèmes de réplication. L'implantation classique de la Diffusion Totalement Ordonnée utilise le Consensus [CHD96] entre les processus pour assurer ses propriétés, ce qui représente un coût assez élevé quand le nombre de participants augmente, soit en nombre de messages échangés, soit en temps de terminaison.

Cependant, il existent des techniques alternatives à celle de Chandra, et qui peuvent bénéficier d'un découpage du réseau afin de fournir un service fiable et efficace. Associées aux techniques de découverte de topologie présentées dans les chapitres précédents, ces techniques deviennent une alternative intéressante aux techniques traditionnelles, comme présenté dans les pages suivantes.

A.1 Diffusion Totalement Ordonnée

L'opération de Diffusion Totalement Ordonnée consiste en assurer que les messages sont délivrés par tous les processus corrects dans le même ordre, malgré l'occurrence des défaillances. Pour cela, les implantations utilisent normalement l'algorithme de Consensus pour mettre en accord tous les processus par rapport aux messages et à leur ordre. Cette procédure nécessite des nombreux échanges de messages, ce qui rend les implantations de Diffusion Totalement Ordonnée fondées sur le Consensus trop chères quand le nombre de processus augment.

Une alternative possible à l'utilisation du Consensus entre les processus est le protocole RBP (*Reliable Broadcast Protocol* - Protocole de Diffusion Fiable) [CHJ84, CHO01]. Ce protocole, structuré autour d'un anneau logique permet une réduction importante du nombre de messages échangés [WHE95], propriété intéressante pour les systèmes de large échelle.

Malheureusement, le protocole RBP est vulnérable à certains patrons de défaillance. En effet, le protocole considère que des processus corrects quittent et rejoignent un groupe à cause des suspicions incorrectes, mais il n'y a aucune garantie que ces processus seront remis en consistance avec les autres processus. Ce problème, appelé *time-bounded buffering* (bufférisation limité par le temps), est dur à résoudre.

Nous avons proposé *i*RBP [BAR02, BAR03], une nouvelle implantation du protocole RBP que non seulement résout le problème du *time-bounded buffering,* mais aussi permet l'augmentation du nombre de processus sans pour autant perdre en performance.

A.2 Définitions

Nous considérons un système distribué asynchrone où un groupe de processus $\pi(t) = \{p1 ; p2 ; ... ; pn\}$ échangent de messages point à point à travers des canaux de communication *fair-lossy*[1]. Pour simplicité, des primitives de multicast (broadcast inclus) sont composées par plusieurs messages point à point.

Les processus sont sujets à des défaillances par « crash » (panne franche) sans la possibilité de récupération. Ainsi, seulement les processus qui ne tombent pas en panne pendant toute l'exécution du programme sont appelés « correct ». Des détecteurs de défaillance de type *Eventually strong* ($\diamondsuit S$) [CHD96] sont utilisés pour suspecter les processus défaillants.

Le problème de la Diffusion Totalement Ordonnée est défini par quatre propriétés [DEF00] :

- **VALIDITÉ** - Si un processus correct diffuse un message *m* à *Dest(m)*, alors au moins un processus correct appartenant à *Dest(m)* finira par délivrer *m*.
- **ACCORD** - Si un processus correct appartenant à *Dest(m)* délivre le message *m*, alors tous les processus corrects appartenant à *Dest(m)* finiront par délivrer *m*.
- **INTÉGRITÉ** - Pour quelconque message *m*, chaque processus correct *p* délivre *m* une seule fois, et seulement si (1) *m* fut envoyé par *sender(m)*, et (2) *p* est un processus appartenant à *Dest(m)*.
- **ORDRE TOTAL** - Si deux processus corrects *p* et *q* délivrent les messages *m* et *m'*, alors *p* délivre *m* avant *m'* si et seulement si *q* délivre *m* avant *m'*.

A.3 Le protocole RBP

RBP utilise la stratégie appelé *moving sequencer* (séquenceur changeant) [DEF00] : seulement un processus à la fois peut définir l'ordre de délivrance des messages, et le rôle de *séquenceur* est transmis entre les processus. RBP peut être facilement implanté en utilisant un algorithme de passage de jeton : le processus que détient le jeton est le seul qui peut attribuer des numéros de séquence aux messages. Une fois que le message (ou le groupe de messages [WHE95]) est ordonné, le jeton est passé au prochain processus.

Quand un processus reçoit le jeton, il l'*accepte* si et seulement s'il détient tous les messages préalablement ordonnés. Si c'est le cas, ce processus attribue un numéro de séquence à un nouveau message et l'envoie au groupe (ce message sert d'acquittement par rapport au séquenceur précédent). Par contre, si le processus détecte qu'il n'a pas tous les messages précédents, il envoie un message « d'acquittement négatif » pour demander explicitement la retransmission des messages manquants. De ce fait, si les canaux ne perdent pas trop de messages, le protocole peut efficacement délivrer des messages avec un minimum d'échange de messages de contrôle [WHE95].

Malheureusement, l'utilisation des liens *fair-lossy* présente quelques problèmes : les processus ne peuvent pas délivrer les messages immédiatement une fois qu'une simple panne peut causer la

[1] **Fair-lossy** - si un processus p envoi un message *m* à *q* un nombre infini de fois et *q* est correct, alors *q* reçoit *m* de *p* un nombre infini de fois.

violation de la propriété *Accord*. Au contraire, les processus doivent attendre que le jeton soit passé entre un certain nombre de processus avant délivre les messages. Si k est le nombre de fois que le jeton doit être passé avant délivrer un message, on appelle ça k-résilience. Si nous prenons par exemple la définition originelle de RBP, les auteurs suggéraient que le jeton soit passé entre tous les processus, la n-résilience (parfois appelé « résilience totale »).

A.3.1 Limitations de RBP

La tolérance aux fautes est essentielle pour RBP, car une simple défaillance peut bloquer le passage du jeton. Ainsi, à l'occurrence d'une défaillance, la liste de processus doit être refaite pour enlever le processus suspect, une procédure appelée « Phase de Réforme » (*Reformation Phase*)[CHJ84].

Pendant la Phase de Réforme, chaque processus que détecte la défaillance contacte les autres processus pour initier une nouvelle liste. Ce processus propose alors une nouvelle liste de processus composé pour tous les processus qu'ont répondu à son appel. Un protocole de validation à trois phases (*3-Phase Commit Protocol*) coordonné par le processus qui a détecté la défaillance [MAX01] sert à l'adoption de la nouvelle liste, qui doit aussi être composée de la majorité des processus.

Malheureusement, la détection se fait d'une façon autonome pour chaque processus, et par conséquent, plusieurs processus peuvent essayer de construire une nouvelle liste. Dans ce cas-là, des processus corrects peuvent être exclus d'une liste seulement parce qu'ils sont à la racine d'une liste concurrente. Dans la définition originelle de RBP [CHJ84], les processus exclus peuvent rejoindre la liste majoritaire au moment où une nouvelle phase de réforme sera initiée. Nous avons observé que cette hypothèse n'est pas suffisante pour garantir la propriété d'*Accord*, occasionnant l'inconsistance entre les processus. En effet, l'hypothèse de RBP peut causer le problème appelé ***time-bounded buffering*** (bufférisation limitée par le temps)[CHB02], où un processus p ne peut pas effacer un message m de sa mémoire à moins que tous les processus en *Dest(m)* ont reçu m ou sont défaillants.

A.4 *i*RBP, une nouvelle approche pour le protocole RBP

Le problème du *time-bounded buffering* représente une grande faiblesse du protocole RBP, mais heureusement ce problème a déjà été étudié dans le contexte du modèle de réplication *Primary-Backup* avec vues synchrones [CHO01]. Deux techniques utilisées pour contrer le problème de la *bufférisation limitée par le temps* se sont révélées facilement adaptables à la structure du protocole RBP. Ces techniques, présentées dans les sections suivantes, font partie de notre nouvelle implantation du protocole RBP appelé *i*RBP, pour *improved* RBP.

A.4.1 Program-Controlled Crash

Dans le travail de Charron-Bost *et al.* [CHB02] il est démontré qu'aucune implantation de Diffusion Fiable (un problème moins dur que la Diffusion Totalement Ordonnée [JAL94]) avec des liens *fair-lossy* peut résoudre le problème de la *bufférisation limitée par le temps* à moins qu'elle utilise un détecteur de défaillance de type *Perfect* (P) [CHD96]. Comme les détecteurs de défaillance de type P ne sont pas réalisables sur des environnements asynchrones, une solution pratique est l'utilisation de *program-controlled crash* (pannes contrôlées par le programme). La technique de *program-controlled crash* donne aux processus le pouvoir de « tuer » d'autres processus (ou les forcer à commettre un suicide), évitant alors le problème de la *bufférisation limitée par le temps* (si tous les processus suspects sont défaillants, alors le message m peut être effacé de la mémoire en toute sécurité).

L'utilisation conjointe de *program-controlled crash* et de du contrôle d'appartenance (*membership*) n'est pas une technique nouvelle (on l'utilisait déjà pour dans le système Isis [BIR96, CHO01]), mais n'est pas très populaire à cause de son coût et de son impact sur la résilience du système. En

effet, à chaque fois qu'un processus q est « tué » à cause d'une suspicion incorrecte, une nouvelle liste d'appartenance est créée pour mettre à jour la vue du système, ce qui a un coût important. Pour éviter que cela se reproduise trop souvent, normalement les détecteurs de défaillance utilisent des temps d'attente trop conservateurs, réduisant considérablement la réactivité des systèmes face à une vraie défaillance.

A.4.2 Deux couches de vues synchrones

Dans les architectures typiques d'appartenance de groupe, résoudre efficacement à la fois les problèmes de la *bufférisation limitée par le temps* et de réactivité n'est pas possible, car les deux aspects sont liés au même schéma de détection de défaillances. Pour contourner cette situation et exploiter au mieux les deux perspectives, Charron-Bost [CHB02] a proposé l'utilisation de deux couches d'appartenance.

Dans son modèle, les *membership views* (appelés simplement *views*) sont identiques aux vues du modèle classique de vues synchrones, alors que les *ranking views* (ou *rk-views*) sont des vues « temporaires » utilisées entre deux changements des listes d'appartenance. Si on représente les *membership views* par v_0, v_1, ..., alors les *rk-views* entre v_i et v_{i+1} sont représentées par $v_i{}^0$, $v_i{}^1$, ..., $v_i{}^{last}$. La liste d'appartenance de tous les *ranking views* $v_i{}^0$, ..., $v_i{}^{last-1}$ est la même que la liste d'appartenance de v_i, et différent seulement sur l'ordre des processus dans les listes. Par exemple, $v_i = v_i{}^0 = \{p,q,r\}$, $v_i{}^1 = \{q,r,p\}$, etc. Comme toutes les *rk*-views contiennent les mêmes processus, l'utilisation de *program-controlled crash* n'est pas nécessaire, au contraire des *membership views*.

Avec cette stratégie en deux couches, nous pouvons résoudre le dilemme entre réactivité et coût du *program-controlled crash*. Les *membership views* sont induits par des détecteurs de défaillance avec des temps d'attente conservateurs, alors que les *rk-views* sont induits par des détecteurs de défaillance avec des temps d'attente beaucoup plus agressifs. De cette manière, les *rk-views* contribuent à augmenter la réactivité des applications et éviter les situations de blocage, alors que les *membership views* garantissent la délimitation du temps de stockage des messages.

A.4.3 Contrôle d'appartenance sous *i*RBP

Pour résoudre les problèmes de RBP, nous avons remplacé la *phase de réforme* originelle par un algorithme d'appartenance en deux couches. Nous avons modifié le modèle de représentation des *ranking views* suggéré par Charron-Bost [CHB02] (placer le coordinateur suspect à la fin de la liste) parce que ce modèle n'était pas adapté à notre protocole fondé sur un anneau. Au lieu de ça, nous utilisons une implantation des *rk-views*, où une vue est composée de deux sous-groupes de processus :

{(groupe principal),(groupe secondaire)}

Les messages sont envoyés à tous les processus, mais le jeton est passé seulement entre les membres du groupe principal. Si un de ces processus du groupe principal est suspecté, alors une nouvelle *rk-view* est créée où ce processus suspect est transféré vers le groupe secondaire. Comme le processus suspecté ne participe plus au passage du jeton, il ne bloque pas le protocole, mais comme il fait encore partie de la vue du système, il reçoit tous les messages du groupe et peut aussi envoyer des nouveaux messages à mettre en séquence.

L'algorithme de changement des *membership views* est juste une version de l'algorithme classique d'appartenance [SCH02] qu'inclut la possibilité de *program-controlled crash* (Algorithme 7). L'algorithme pour le changement des *rk*-views, par contre, est plus optimisé, une fois que un changement des *rk-views* ne requiert pas les informations sur les messages délivrés (en effet, les processus nécessitent seulement la liste des processus actifs) ni *program-controlled crash*. Par conséquent, l'Algorithme 8 présente cette version optimisée d'appartenance, adaptée au changement des *rk*-views.

Algorithme 7 Changement des listes d'appartenance avec *program-controlled crash*

Upon suspicion of some process in TL_i by a conservative Failure Detector
RBroadcast (reformation, i)
Upon R-Deliver (reformation, i) by p_k for the first time
1. send $seqQ_k$ to all /* sends the "unstable" list of messages */
2. $\forall$ p_i, wait until receive $seqQ_i$ from p_i or p_i suspected
3. let $initial_k$ be the tuple (TL, $Msgs_k$) s.t.
- TL means {($core_k$),()}, and $core_k$ contains all processes that sent their seqQ
- $Msgs_k$ is the union of the seqQ sets received
4. execute Consensus among TL_i processes, with $initial_k$ as the initial value
5. let (TL, Msg) be the consensus decision
6. stableQ $\leftarrow$ Msg, seqQ $\leftarrow$ {} /* the set Msg becomes stable */
7. if $p_k \in$ TL, then "install" TL as the next view TL_{i+1}
else suicide

Algorithme 8 Changement optimisé des *rk*-views

Upon suspicion of some process in TL_i^j by an aggressive Failure Detector
RBroadcast (rk-view, j)
Upon R-Deliver (rk-view, j) by p_k for the first time
1. send ack to all
2. $\forall$ p_i, wait until receive ack from p_i or p_i suspected
3. let $initial_k$ contain {($core_k$),($external_k$)} s.t.
- $core_k$ is the list of all processes that sent ack
- $external_k$ is the list of all processes that does not sent ack
4. execute Consensus among TL_i^j processes, with $initial_k$ as the initial value
5. let (TL) be the consensus decision
6. if $p_k \in$ TL, then "install" TL as the next *rk*-view TL_i^{j+1}

A.4.4 Niveaux de résilience et topologie

Comme présenté dans la section A.3, RBP [CHJ84] suggère l'utilisation de la résilience totale, pour fournir le plus grand niveau de tolérance aux fautes. Pourtant, la résilience totale induit un grand délai pour la livraison des messages, et représente un coût trop élevé pour des applications à grande échelle. Avec *i*RBP, le niveau de résilience est lié au nombre de processus dans le groupe principal, et par conséquent, le contrôle du nombre de processus dans ce groupe est une technique efficace pour limiter la latence de livraison du protocole.

Les sections suivantes présentent quelques analyses où nous évaluons l'influence du nombre de niveaux de résilience par rapport au nombre de messages échangés et les délais de livraison des messages.

A.5 Analyse de performance

A.5.1 Nombre de messages échangés

Une des caractéristiques du protocole *i*RBP et qui lui rend très attractif pour les systèmes à grande échelle est son utilisation efficace des ressources. En effet, si aucun message est perdu, *i*RBP requiert asymptotiquement seulement deux diffusions par message : le premier, envoyé par la source du message ; et le deuxième, envoyé par le séquenceur, qui au même temps donne le numéro de séquence du message et passe le jeton au prochain séquenceur. Comme résultat, la diffusion fiable d'un message nécessite seulement $2 \times (n-1)$ messages (si on considère que la diffusion transmet $n-1$ messages point à point) échangés entre les processus.

Par comparaison, le protocole de Totalement Ordonnée fondée sur le Consensus [CHD96] peut échanger jusqu'à $(4+2n) \times (n-1)$ messages, et même les implantations plus optimisées du

Consensus, comme par exemple le *Early Consensus* de Schiper [SCH97], envoient encore $O(n^2)$ messages.

A.5.2 Latence, *k*-résilience et scalabilité

Un élément important pour l'analyse des protocoles d'ordre total est la *latence*. Dans le contexte de ces protocoles, la latence est le temps écoulé entre le broadcast d'un message m par le processus source et la première fois que ce message est délivré à l'application. Dans le cas de *i*RBP, la latence dépend du niveau de résilience utilisé, car un message est délivré à l'application seulement quand le niveau de résilience est atteint.

Pour limiter le temps de livraison des messages, nous utilisons la propre structure du protocole d'appartenance en deux couches. En exerçant un contrôle sur le nombre de processus dans le groupe principal, il est possible de limiter la latence même si le nombre total de processus augmente.

Nous avons implanté *i*RBP en Java, utilisant comme support le framework Neko [URB01]. Les expériences furent exécutées sur la grappe **ID/HP i-cluster1**, qui comptait jusqu'à 256 machines interconnectées par un réseau Ethernet 100 Mbps commuté. Les tests ont utilisé 4, 8, 16, 32 et 64 machines (un processus/machine), dont nous comparons la performance de *i*RBP sous différents niveaux de résilience. Les résultats, illustrés dans la Figure A.1 présentent les temps de livraison moyens pour des messages qu'arrivent selon une distribution de Poisson à un taux de 10 messages par seconde. Pour une question d'uniformité, nous considérons qu'un processus ordonne un message à la fois, malgré la possibilité de mettre en séquence plusieurs messages à la fois.

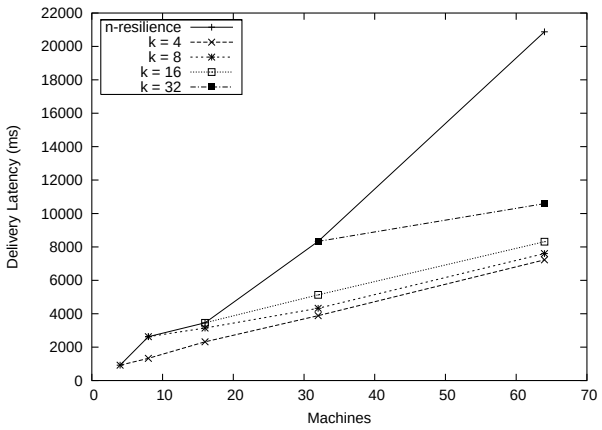


FIG. A.1 *Influence des k-résilience sur la latence de livraison des messages*

Nous observons que dans le modèle originel de RBP (résilience totale) la latence augmente presque linéairement avec le nombre de processus dans le groupe. Par contre, si on limite le nombre de processus à l'intérieur du groupe principal, les valeurs de latence sont bien plus réduites.

En résumé, le choix d'un nombre petit de membres pour le groupe principal a permis au protocole de délivrer les messages avec des niveaux de latence similaires à ceux des techniques conventionnelles (en effet, une Diffusion Totalement Ordonnée fondée sur le Consensus nécessite au moins six étapes de communication), alors que le nombre de messages échangés est fortement réduit par rapport aux mêmes techniques. De plus, les implantations peuvent bénéficier d'avantages comme les primitives bas-niveau de type Ethernet-Broadcast ou IP-Multicast, qui ne sont pas applicables à la Diffusion Totalement Ordonnée fondée sur le Consensus, ou alors hiérarchiser les communications, de manière à ce que les processus d'une grappe ne se communiquent qu'avec son coordinateur, évitant ainsi la diffusion des messages entre tous les processus et la surcharge des liens.

A.6 Discussion

Les applications plus récentes de RBP [MAX01] visent l'interconnexion des systèmes à une échelle globale, où un certain nombre de machines fiables assurent la résilience du service. Ici le contrôle des membres du groupe principal est une nécessité et une façon d'hiérarchiser les communications par rapport à la fiabilité des membres.

Dans le contexte d'une grille de calcul, il est facilement concevable une organisation du groupe où le groupe principal est composé d'un représentant de chaque grappe, qui assure la rediffusion des messages manquants à l'intérieur de ses grappes. Ainsi, les outils de découverte de topologie peuvent fournir les informations nécessaires à la bonne répartition des processus et même des indices pour estimer le degré de fiabilité de certaines machines candidates au groupe principal et le réglage des *timeouts*.

Annexe B

Paramètres d'Interconnexion pLogP entre les Grappes de GRID5000

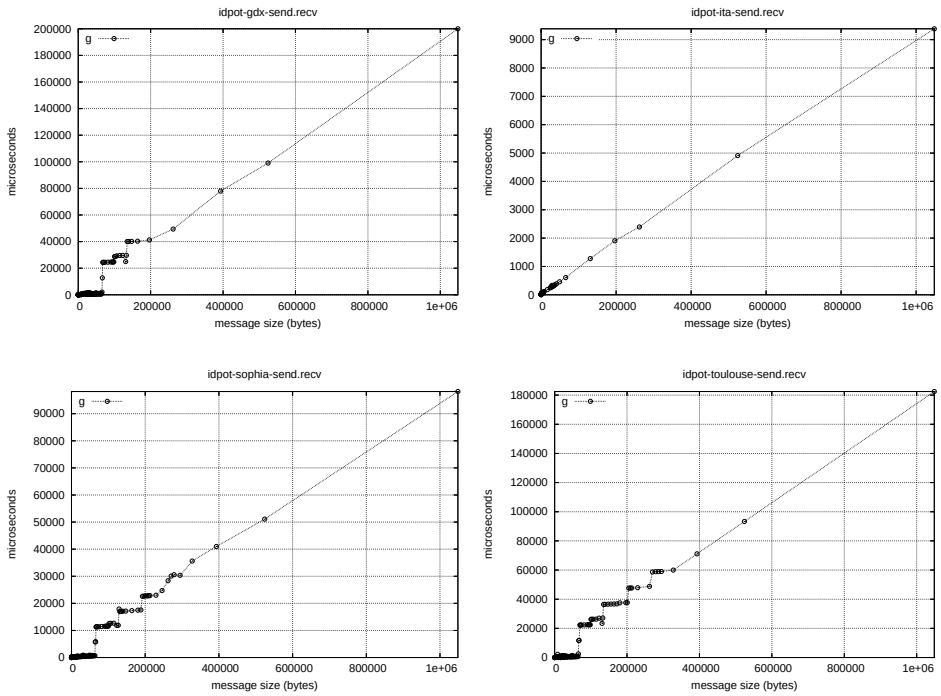


FIG. B.1 *Valeurs du gap mesuré à partir de la grappe IDPOT*

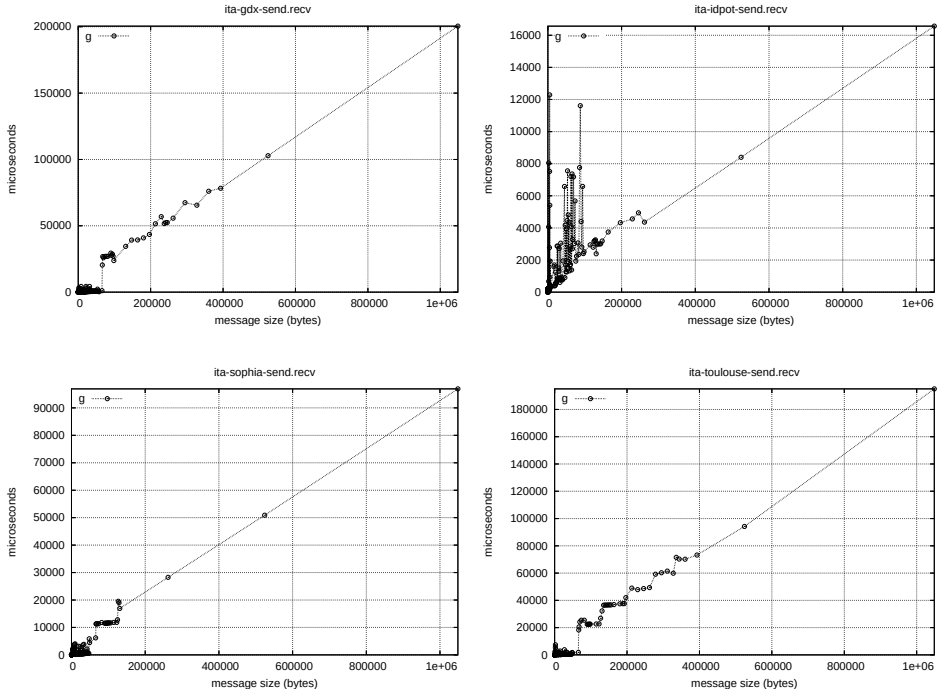


FIG. B.2 *Valeurs du gap mesuré à partir de la grappe* icluster-2

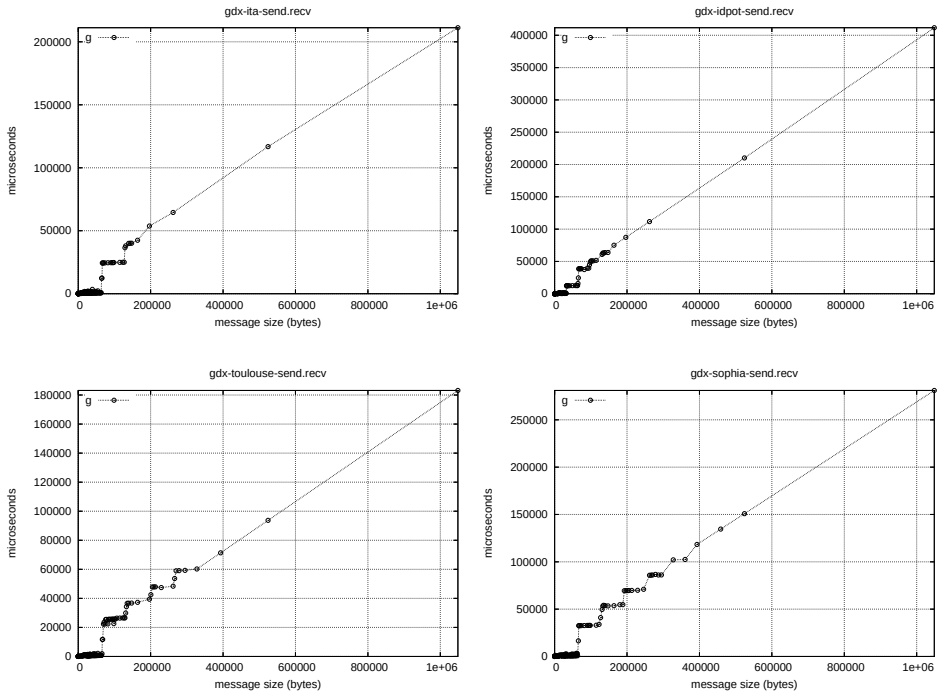


FIG. B.3 *Valeurs du gap mesuré à partir de la grappe Grid eXplorer (GdX)*

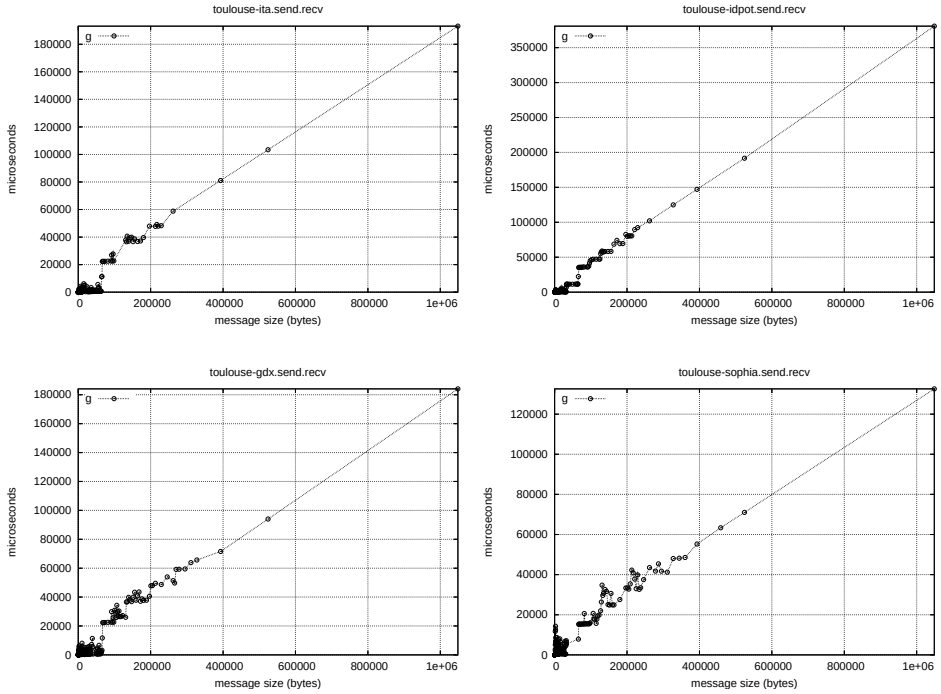


FIG. B.4 *Valeurs du gap mesuré à partir de la grappe de Toulouse*

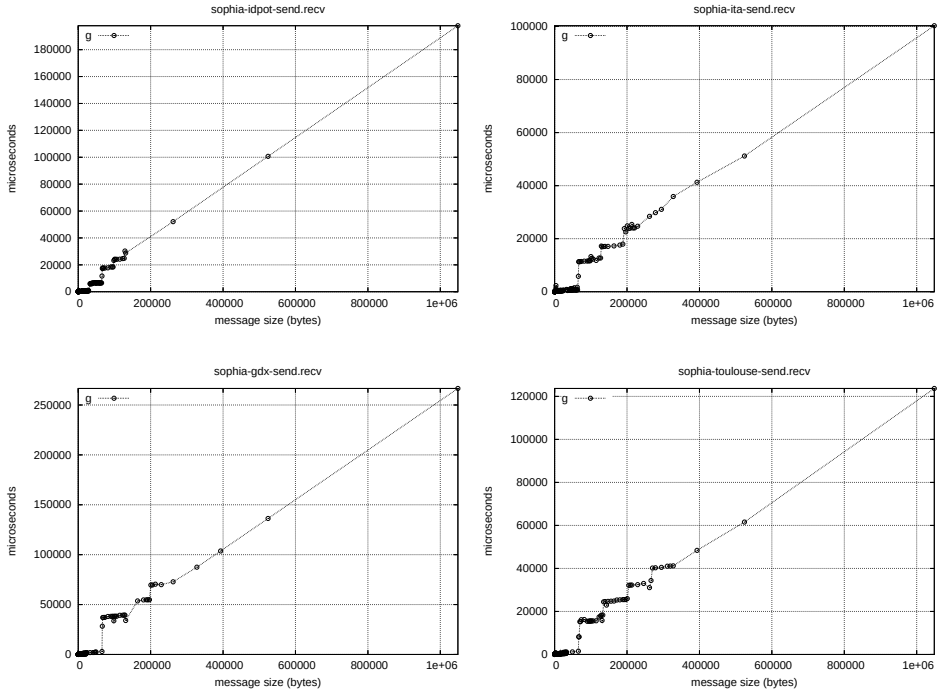


FIG. B.5 *Valeurs du gap mesuré à partir de la grappe de Sophia-Antipolis*

Bibliographie

[ADV93] Vikram Adve, ”Analysing the Behavior and Performance of Parallel Programs”. PhD Thesis, University of Wisconsin, Computer Sciences Department, 1993.

[ALE95] Albert Alexandrov, Mihai Ionescu, Klaus Schauser, Chris Scheiman, "LogGP : Incorporating Long Messages into the LogP model - One step closer towards a realistic model for parallel computation". 7th Annual Symposium on Parallel Algorithms and Architecture (SPAA'95), 1995.

[BAN98] Mohammad Banikazemi, Vijay Moorthy, Dhabaleswar K. Panda, "Efficient Collective Communication on Heterogeneous Networks of Workstations". International Conference on Parallel Processing (ICPP'98), pp. 460-467, 1998.

[BAO94] Amotz Bar-Noy, Shlomo Kipnis, "Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems". Math. Systems Theory, vol. 27, no. 5, pp. 431-452, 1994.

[BAR02] Luiz Barchet-Estefanel, "Analysing RBP, a Total Order Broadcast Protocol for Unreliable Networks". Technical Report, IC-EPFL - Switzerland, 2002.

[BAR03] Luiz Barchet-Estefanel, "iRBP - A Fault Tolerant Total Order Broadcast for Large Scale Systems”. In *Proc. of the EuroPar 2003*, LNCS Vol. 2790, pp. 632-639, 2003.

[BAR04a] Luiz Barchet-Estefanel, Gregory Mounié, "Fast Tuning of Intra-cluster Collective Communications”. In *Proc. of the Euro PVM/MPI 200*4, LNCS Vol. 3241, pp. 28-35, 2004.

[BAR04b] Luiz Barchet-Estefanel, Gregory Mounié, "Identifying Logical Homogeneous Clusters for Efficient Wide-Area Communication". In *Proc. of the Euro PVM/MPI 200*4, LNCS Vol. 3241, pp. 319-326, 2004.

[BAR04c] Luiz Barchet-Estefanel, Gregory Mounié, "Performance Characterisation of Intra-Cluster Collective Communications". In *Proc. of the 16th Symposium on Computer Architecture and High Performance Computing* (SBAC-PAD 2004) pp. 254-261, 2004.

[BAR05a] Luiz Barchet-Estefanel, Gregory Mounié, "Prédiction de Performances pour les Communications Collectives". In *Proceedings du 16ème Rencontre Francophone du Parallélisme (RenPar'16),* pp. 101-112, 2005.

[BAR05b] Luiz Barchet-Steffenel, Gregory Mounié, "Total Exchange Performance Modelling under Network Contention". In *Proceedings of the 6th International Conference on Parallel Processing and Applied Mathematics, to appear*, 2005.

[BAT96] Mike Barnett, David Payne, Robert van de Geijn, Jerrel Watts, "Broadcasting on meshes with wormhole routing". *Journal of Parallel and Distributed Computing*, 35(2), pp. 111-122, 1996.

[BEA04a] Olivier Beaumont, Loris Marchal, "Pipelining Broadcasts on Heterogeneous Platforms under the One-Port Model". Technical Report 2004-32, LIP, ENS Lyon, France, 2004.

[BEA04b] Olivier Beaumont, Arnaud Legrand, Loris Marchal et Yves Robert, "Pipelining Broadcasts on Heterogeneous Platforms". International Parallel and Distributed Processing Symposium (IPDPS 2004), 2004.

[BEA04c] Olivier Beaumont, Loris Marchal, Yves Robert. "Broadcast Trees for Heterogeneous Platforms", Technical Report 2004-46, LIP, ENS Lyon, France, 2004.

[BEA05a] Olivier Beaumont, Arnaud Legrand, Loris Marchal et Yves Robert, "Pipelining Broadcasts on Heterogeneous Platforms". IEEE Transactions on Parallel and Distributed Systems, Vol. 16, No. 4, pp. 300-313, 2005.

[BEA05b] Olivier Beaumont, Loris Marchal et Yves Robert, "Broadcast Trees for Heterogeneous Platforms". International Parallel and Distributed Processing Symposium (IPDPS 2005), 2005.

[BEJ03] Yigal Bejerano, Yuri Breitbart, Minos Garofalakis, Rajeev Rastogi, "Physical Topology Discovery for Large Multi-Subnet Networks". IEEE Infocom 2003.

[BEN03] Gregory D. Benson, Cho-Wai Chu, Qing Huang, Sadik G. Caglar, "A Comparison of MPICH Allgather Algorithms on Switched Networks". Euro PVM/MPI 2003, LNCS 2840, pp. 335-343, 2003.

[BER98] Massimo Bernaschi, Giulio Iannello, "Collective Communication Operations : Experimental Results vs. Theory". Concurrency : Practice and Experience, vol. 10, No. 5, pp. 359-386, April 1998.

[BER03] Massimo Bernaschi, Giulio Iannello, Mario Lauria, "Efficient implementation of reduce-scatter in MPI". Journal of Systems Architecture 49, pp. 89-108, 2003.

[BHA98] Prashanth Bhat, Viktor Prasanna, C.S. Raghavendra, "Adaptive Communication Algorithms for Distributed Heterogeneous Systems". Proceedings of the 7th IEEE Intnl. Symposium on High Performance Distributed Computing (HPDC 1998), 1998.

[BHA99] Prashanth Bhat, Viktor Prasanna, C.S. Raghavendra, "Adaptive communication algorithms for distributed heterogeneous systems". Journal of Parallel and Distributed Computing 59 (1999) 252-279.

[BHA03] Prashanth Bhat, C.S. Raharendra, Viktor Prasanna, "Efficient Collective Communication in Distributed Heterogeneous Systems". Journal of Parallel and Distributed Computing, No. 63, Elsevier Science. pp. 251-263, 2003.

[BIE00] Andy Bierman, Kendall Jones, "Physical Topology MIB". RFC2922, september 2000.

[BIR96] Kenneth Birman, "Building Secure and Reliable Network Applications". Manning Books, 1996.

[BOD95] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, et Wen-King Su, "Myrinet : A Gigabit-per-Second Local Area Network". IEEE Micro, 15(1) :29-36, 1995.

[BOE96] Maria Cristina Silva Boeres, ”Versatile Communication Cost Modelling for Multicomputer Task Schedulling Heuristics”. PhD Thesis, University of Edinbourgh, 1996.

[BOS02] George Bosilca, Aurélien Bouteiller, Franck Cappello, Samir Djilali, Gilles Fédak, Cécile Germain, Thomas Hérault, Pierre Lemarinier, Oleg Lodygensky, Frédéric Magniette, Vincent Néri, Anton Selikhov, "MPICH-V : Toward a Scalable Fault Tolerant MPI for Volatile Nodes". In proceedings of The IEEE/ACM SC2002 Conference, Baltimore USA, November 2002.

[BRE00] Yuri Breitbart, Minos Garofalakis, Cliff Martin, Rajeev Rastogi, S. Seshadri, Avi Silberschatz, "Topology Discovery in Heterogeneous IP Networks". Proceedings IEEE INFOCOM 2000, The Conference on Computer Communications, pp. 265-274, 2000.

[BRU96] Jehoshua Bruck, Luc De Coster, Natalie Dewulf, Ching-Tien Ho, Rudy Lauwereins, "On the Design and Implementation of Broadcast and Global Combine Operations Using the Postal Model". IEEE Transactions on Parallel and Distributed Systems, Vol. 7, No. 3, Mars 1996.

[BRU97a] Jehoshua Bruck, Danny Dolev, Ching-Tien Ho, Marcel-Catalin Rosu, H. Raymond Strong, "Efficient Message Passing Interface (MPI) for Parallel Computing on Clusters of Workstations". Journal of Parallel and Distributed Computing 40(1) :19-34, 1997.

[BRU97b] Jehoshua Bruck, Ching-Tien Ho, Shlomo Kipnis, Eli Upfal, Derrick Weathersby, "Efficient algorithms for all-to-all communications in multiport message-passing systems". IEEE Transactions on Parallel and Distributed Systems, 8(11) :1143-1156, November 1997.

[BUR02] Mathijs den Burger, Thilo Kielmann, Henri Bal, "TopoMon : a monitoring tool for Grid network topology". International Conference on Computational Science'02. Springer-Verlag, LNCS Vol. 2330, pp. 558-567, 2002.

[BUR05] Mathijs den Burger, Thilo Kielmann, Henri Bal, "Balanced Multicasting : High-throughput Communication for Grid Applications". Supercomputing 2005 (SC05), Seattle, November 12-18, 2005.

[CAL95] Christophe Calvin, S. Perennes, Denis Trystram. "All-to-all Broadcast in Torus with Wormhole-Like Routing". IEEE Symposium on Parallel and Distributeed Processing, pp. 130-137, 1995.

[CAP03] Nicolas Capit, Georges Da Costa, Guillaume Huard, Cyrille Martin, Grégory Mounié, Pierre Neyron, Olivier Richard, "Expériences autour d'une nouvelle approche de conception d'un gestionnaire de travaux pour grappe". 3ème Conférence Française sur les Systèmes d'Exploitation (CFSE), 2003.

[CHA04] Ernie W. Chan, Marcel F. Heimlich, Avi Purkayastha, Robert A. van de Geijn, "On Optimizing Collective Communication", IEEE Cluster, 2004.

[CHB02] Bernardette Charron-Bost, Xavier Défago, André Schiper, "Broadcasting Messages in Fault-Tolerant Distributed Systems : the benefit of handling input-triggered and output-triggered suspicions differently". Proceedings of the 21stInt'l Symposium on Reliable Distributed Systems, Osaka, Japan. 2002.

[CHD96] Tushar Deepak Chandra, Sam Toueg, "Unreliable failure detectors for reliable distributed systems". Journal of the ACM, 43(2). ACM Press New York, NY, USA, pp. 225-267, 1996.

[CHJ84] Jo-Mei Chang, Nicholas Maxemchuk, "Reliable Broadcast Protocols". ACM Trans. on Computer Systems, 2(3). ACM Press New York, NY, USA, pp. 251-273, 1984

[CHE96] Ming-Syan Chen, Jeng-Chun Chen, et Philip S. Yu, "On General Results for All-to-All Broadcast". IEEE Transactions on Parallel and Distributed Systems, Vol. 7, No. 4, pp. 363-370, Avril 1996.

[CHO01] Gregory Chockler, Idith Keidar, Roman Vitenberg, "Group Communication Specifications : a comprehensive study". ACM Computing Surveys, 33(4). ACM Press New York, NY, USA, pp. 427-469, 2001.

[CHR99] Christina Christara, Xiaoliang Ding, Ken Jackson, "An efficient transposition algorithm for distributed memory computers". In *Proc. High Performance Computing Systems and Applications*, pp. 349-368, 1999.

[CLE96] Mark Clement, Michael Steed, Phyllis Crandall, "Network performance modelling for PM clusters". In *Proceedings of Supercomputing*, 1996.

[CUL96a] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, Thorsten von Eicken, "LogP - A practical model of parallel computing". *Communication of the ACM*, 39(11), pp. 78-85, 1996.

[CUL96b] David Culler, Lok Tin Liu, Richard Martin, Chad Yoshikawa, "Assessing Fast Network Interfaces". *IEEE Micro*, 16(1), pp. 35-43, 1996.

[DEF00] Xavier Défago, "Agreement-related Problems : from semi-passive replication to totally ordered broadcasts". PhD Thesis, EPFL - Switzerland, 2000.

[DIN01] Peter Dinda, Thomas Gross, Roger Karrer, Bruce Lowekamp, Nancy Miller, Peter Steenkiste, Dean Sutherland, "The Architecture of the Remos System". In 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10), August 2001.

[ELN96] Mootaz Elnozahy, Lorenzo Alvisi, Yi-Min Wang, David B. Johnson, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems". Technical Report CMPU-CS-96-181, Carnegie Mellon University, 1996.

[FAG03] Graham E. Fagg, Edgar Gabriel, Zizhong Chen, Thara Angskun, George Bosilca, Antonin Bukovski, Jack Dongarra, "Fault tolerant communication library and applications for high performance". In Los Alamos Computer Science Institute Symposium, 2003.

[FAR05] Ahmad Faraj, Xin Yuan, "Message Scheduling for All-to-all Personalized Communication on Ethernet Switched Clusters". 19th IEEE International Parallel & Distributed Processing Symposium (IPDPS), Denver, Colorado, April 4-8, 2005.

[FIG00] Silvia M. Figueira, "Using a Hypercube Algorithm for Broadcasting in Internet-Based Clusters". International Conference on Parallel and Distributed Processing Techniques and Applications, 2000.

[FIG02] Silvia M. Figueira, "Improving Binomial Trees for Broadcasting in Local Networks of Workstations". VECPAR 2002, Portugal, 2002.

[FIG04] Silvia M. Figueira, Christine Mendes., "Dynamically Adaptive Binomial Trees for Broadcasting in Heterogeneous Networks of Workstations". VECPAR 2004, LNCS 3402, pp 480-495, 2004.

[FOL98] Gianluigi Folino, Giandomenico Spezzano, Domenico Talia, "Performance Evaluation and Modeling of MPI Communications on the Meiko CS-2", Proc. of HPCN Europe'98, LNCS 1401, Springer-Verlag, pp. 932-936, April 1998.

[FOR78] Stephen Fortune, James Wyllie, "Parallelism in random access machines". In Proc. of 10th Annual ACM Symposium on Theory of Computing, pages 114118. ACM Press, 1978.

[FOS99] Ian Foster, Karl Kesselman (Eds.). "The Grid : Blueprint for a New Computing Infrastructure". Morgan Kaufmann, 1999.

[FRA01] Paul Francis, Siguh Jamin, Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, et Lixia Zhang. "IDMaps : A Global Internet Host Distance Estimation Service". IEEE/ACM Transactions on Networking, oct 2001. Available at http ://citeseer.nj.nec.com/francis01idmaps.html.

[GAB04] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, Timothy S. Woodall, "Open MPI : Goals, Concept and Design of a Next Generation MPI Implementation". Euro PVM/MPI 2004, LNCS 3241, pp 97-104, 2004.

[GEI94] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, Vaidy Sunderam, "PVM : Parallel Virtual Machine". MIT press, 1994.

[GOL98] Alfredo Goldman, Denis Trystram, Joseph G. Peters, "Exchange of Messages of Different Sizes". IRREGULAR 1998 : pp. 194-205, 1998.

[GOL02] Alfredo Goldman, "Scalable Algorithms for Complete Exchange on Multi-Cluster Networks". CCGRID 2002, pp. 286-287, 2002.

[GOL05] Alfredo Goldman, Denis Trystram, Joseph G. Peters, "Exchange of Messages of Different Sizes". Journal of Parallel and Distributed Computing, Elsevier Science, to appear. 2005.

[GRI05] Projet GRID5000, http ://www.grid5000.fr, 2005.

[GRO03] Duncan Grove, "Performance Modelling of Message-Passing Parallel Programs", PhD. Thesis, University of Adelaide, 2003.

[HAN00] Jizhong Han, Chengde Han, "Efficient All-to-All Broadcast in Switch-based Networks with Irregular Topology". 4th International Conference on High-Performance Computing in the Asia-Pacific Region (HPC'2000), pp. 112-114, 2000.

[HAT00] Jun-ichi Hatta, Susumu Shibusawa, "Scheduling Algorithms for Efficient Gather Operations in Distributed Heterogeneous Systems". ICPP Workshops, pp. 173-180, 2000.

[HEN88] Debra Hensgen, Raphael Finkel, Udi Manber, "Two algorithms for Barrier Synchronization". International Journal of Parallel Programming, Vol. 17, No. 1, 1988.

[HOK94] R.W. Hockney, "The Communication Challenge for MPP : Intel Paragon and Meiko CS-2". Parallel Computing, North-Holland, vol. 20, pp. 389-398, 1994.

[HUS99] Lars Paul Huse, "Collective Communication on Dedicated Clusters of Workstations". Proceedings of the 6th European PVM/MPI Users' Group Meeting, Barcelona, Spain, pp. 469-476, 1999.

[IAN97] Giulio Iannello, "Efficient Algorithms for the Reduce-Scatter Operation in LogGP". IEEE Transactions on Parallel and Distributed Systems, v.8 n.9, p.970-982, September 1997.

[IAN98] Giulio Iannello, Mario Lauria et Stefano Mercolino, "Cross-Platform Analysis of Fast Messages for Myrinet". In *Workshop CANPC'98*, LNCS Vol. 1362, pp. 217-231, 1998.

[IMPI] Interoperable MPI Web page. *http ://impi.nist.gov*

[JAC99] Matthew Jacunski, P. Sadayappan, Dhabaleswar K. Panda, "All-to-All Broadcast on Switch-Based Clusters of Workstations". International Parallel Processing Symposium (IPPS'99), pp. 325-329, 1999.

[JAI91] Raj Jain, "Art of Computer Systems Performance Analysis Techniques for Experimental Design Measurements Simulation and Modeling". Wiley Computer Publishing, John Wiley and Sons Inc., 1991.

[JAL94] Pankaj Jalote, "Fault Tolerance in Distributed Systems". Prentice-Hall, 1994.

[JUH04] Sandor Juhász, Ferenc Kovács, "Asynchronous Distributed Broadcast in Cluster Environments". Euro PVM/MPI 2004, LNCS 3241, pp 164-172, 2004.

[KAL03] Laxmikant V. Kalé, Sameer Kumar, Krishnan Varadarajan, "A Framework for Collective Personalized Communication". Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03), 2003.

[KAP93] Richard M. Karp, Abhijit Sahay, Eunice Santos, Klaus E. Schauser, "Optimal Broadcast and Summation in the LogP Model". ACM Symposium on Parallel Algorithms and Architectures (SPAA'93), pp. 142-153, 1993.

[KAR00] Nicholas T. Karonis, Bronis Supinski, Ian Foster, William Gropp, Edwin Lusk, John Bresnahan, "Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance". In : 14th International Conference on Parallel and Distributed Processing Symposium. IEEE Computer Society pp. 377-384, 2000.

[KAR02] Nicholas T. Karonis, Ian Foster, Bronis Supinski, William Gropp, Edwin Lusk, Sébastien Lacour, "A Multilevel Approach to Topology-Aware Collective Operations in Computational Grids". Technical report ANL/MCS-P948-0402, Mathematics and Computer Science Division, Argonne National Laboratory, 2002.

[KAR03] Nicholas T. Karonis, Brian Toonen, Ian Foster, "MPICH-G2 : A Grid-enabled implementation of the Message Passing Interface". Journal of Parallel and Distributed Computing, No. 63, Elsevier Science, pp. 551-563, 2003.

[KES97] Ram Kesavan, Dhabaleswar K. Panda, "Optimal Multicast with Packetization and Network Interface Support". International Conference on Parallel Processing (ICPP'97), 1997.

[KIE99a] Thilo Kielmann, Rutger Hofman, Henri Bal, Aske Plaat, Raoul Bhoedjang, "MPI's Reduction Operations in ClusteredWide Area Systems". Proc. Message Passing Interface Developer's and User's Conference (MPIDC'99), pp. 43-52, Atlanta, GA, March 10-12, 1999.

[KIE99b] Thilo Kielmann, Rutger Hofman, Henri Bal, Aske Plaat, Raoul Bhoedjang, "MagPIe : MPI's Collective Communication Operations for Clustered Wide Area Systems". In : 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM Press, pp. 131-140, 1999.

[KIE00] Thilo Kielmann, Henri Bal, Kees Verstoep, "Fast Measurement of LogP Parameters for Message Passing Platforms". In *4th Workshop on Runtime Systems for Parallel Programming*, LNCS Vol. 1800, pp. 1176-1183, 2000.

[KIE01] Thilo Kielmann, Henri Bal, Sergey Gorlatch, Kees Verstoep, Rutger Hofman, "Network Performance-aware Collective Communication for Clustered Wide Area Systems". Parallel Computing, Vol. 27, No. 11, Elsevier Science. pp. 1431-1456, 2001.

[LAB96] Jesús Labarta, Sergi Girona, Vincent Pillet, Toni Cortes, Luis Gregoris, "DiP : A parallel program development environment". In *Proc. of the 2ndEuro-Par Conference*, vol II, pp. 665-674, 1996.

[LAC01] Sébastien Lacour, Nicholas T. Karonis, Ian Foster, "MPICH-G2 Collective Operations Performance evaluation, optimizations". Technical report ANL, Mathematics and Computer Science Division, Argonne National Laboratory, 2001.

[LAM01] LAM-MPI Team, "Performance Issues with LAM/MPI on Linux 2.2.x", http ://www.lam-mpi.org/linux/, 2001.

[LAM03] LAM-MPI Team. LAM/MPI Version 6.5.2. http ://www.lam-mpi.org/, 2003.

[LAM04] LAM-MPI Team, LAM/MPI Version 7, http ://www.lam-mpi.org/, 2004.

[LEG03] Arnaud Legrand, Frédéric Mazoit, Martin Quinson, "An Application-Level Network Mapper". Research Report No 2003-09, LIP - Lyon, 2003.

[LEG04] Arnaud Legrand, Martin Quinson, "Automatic deployment of the Network Weather Service using the Effective Network View". In : High-Performance Grid Computing Workshop (associated to IPDPS'04), IEEE Computer Society, 2004.

[LI01] Chun-Hua Li, Xing-Ming Zhou, "GIAS : A Global Information Acquisition System in a Cluster Environment". International Workshop on Advanced Parallel Processing Technologies, Ilmenau, Germany, 2001.

[LIU00a] Pangfeng Liu, Da-Wei Wang, "Reduction optimization in heterogeneous cluster environments". Proc. of 14th IPDPS, pp. 477-482, 2000.

[Liu00b] Pangfeng Liu, Tzu-Hao Sheng, "Broadcast scheduling optimization for heterogeneous cluster systems" Proc. of 12th SPAA, pp. 129-136, 2000.

[Liu02] Pangfeng Liu, "Broadcast Scheduling Optimization for Heterogeneous Cluster Systems". Journal of Algorithms 42(1) : 135-152, 2002.

[Liu03] Pangfeng Liu, Da-Wei Wang, Yi-Heng Guo, "An Approximation Algorithm for Broadcast Scheduling in Heterogeneous Clusters". Real-Time and Embedded Computing Systems and Applications, 9th International Conference (RTCSA 2003), LNCS 2968, pp. 38-52, 2004.

[LIW96] Wenheng Liu, Cho-Li Wang, Viktor K. Prasanna, "Portable and Scalable Algorithms for Irregular All-to-All Communication". In Proceedings of the 16th ICDCS, pp. 428-435, 1996.

[LON00] Josip Loncaric, "Linux TCP patches to improve acknowledgement policy", http ://research.nianet.org/ ~josip/LinuxTCP-patches.html, 2000.

[LOW96] Bruce Lowekamp, Adam Beguelin, "ECO : Efficient Collective Operations for communication on heterogeneous networks". In : 10th International Parallel Processing Symposium. pp. 399-405, 1996.

[LOW00] Bruce Lowekamp, "Discovery and Application of Network Information". PhD Thesis, Carnegie Mellon University, 2000.

[MAR04] Corine Marchand, "Dimensionnement et mise au point d'algorithmes distribués dans un environnement fortement variable : expérimentation dans le contexte des pico-réseaux". Thèse de doctorat, Institut National Polytechnique de Grenoble, Décembre 2004.

[MAT05] Gabriel Mateescu, "A method for MPI Broadcast in Computational Grids". International Parallel and Distributed Processing Symposium (IPDPS 2005), 2005.

[MAX01] Nicholas Maxemchuk, David Shur, "An Internet Multicast System for the Stock Market". ACM Trans. on Computer Systems, 19(3). ACM Press New York, NY, USA, pp. 384-412, 2001.

[MID04] Edson T. Midorikawa, Helio M. Oliveira, Jean M. Laine, "PEMPIs : A New Metodology for Modeling and Prediction of MPI Programs Performance", in *Proc. of the SBAC-PAD 200*4, IEEE Computer Society, pp. 254-261, 2004.

[MIL98] Zina Ben Miled, José A. B. Fortes, Rudolf Eingenmann, Valerie Taylor, "On the Impleemntation of Broadcast, Scatter and Gather in a Heterogeneous Architecture". IEEE Hawaii International Conference on System Sciences, 1998.

[MOR01] Csaba. A. Moritz, Matthew I. Frank, "LoGPC : Modeling Network Contention in Message-Passing Programs". *IEEE Trans. Parallel and Distributed Systems*, 12(4), pp. 404-415, 2001.

[MPI03] MPICH Team. MPICH Version 1.2.5. http ://www-unix.mcs.anl.gov/mpi/mpich/, 2003.

[NG01] T.S. Eugene Ng, Hui Zhang, "Predicting Internet Network Distance with Coordinates-Based Approaches", 2001. Available at http ://citeseer.nj.nec.com/article/ng01predicting.html.

[NIS03] Rajesh Nishtala, Neil Patel, Kaushal Sanghavi, Kushal Chakrabarti, "Automatic Tuning of Collective Communication Operations in MPI". Rapport de recherche CS262A, University of California, Berkeley, Deécembre 2003.

[OOS02] Fukuhito Ooshita, Susumu Matsumae, Toshimitsu Masuzawa, "Efficient Gather Operation in Heterogeneous Cluster Systems". Proceedings of the 16th Annual International Symposium on High Performance Computing Systems and Applications (HPCS02), 2002.

[PAJ96] Ju-Young L. Park, Hyeong-Ah Choi, Natawut Nupairoj, Lionel M. Ni, "Construction of Optimal Multicast Trees Based on the Parameterised Communication Model". International Conference on Parallel Processing (ICPP 1996), pp 180-187, 1996.

[PAK03] Kyung-Lang Park, Hwang-Jik Lee, Kwang-Won Koh, Oh-Young Kwon, Sung-Yong Park, Hyoung-Woo Park, Shin-Dug Kim, "Dynamic Topology Selection for High Performance MPI in the Grid Environments", Euro PVM/MPI 2003, LNCS 2840, pp. 595-602, 2003.

[PBS] OpenPBS. http ://www.openpbs.org.

[PEJ81] James Lyle Peterson, "Petri Net Theory and Modeling of Systems". Prentice-Hall, Englewood-Cliffs, New Jersey, 1981.

[PET03] Fabrizio Petrini, Juan Fernandez, Eitan Frachtenberg, Salvador Coll, "Scalable Collective Communication on the ASCI Q Machine". Proceedings of the 11 th Symposium on High Performance Interconnects (HOTI'03), 2003.

[PJE05] Jelena Pjesivac-Grbovic, Thara Angskun, George Bosilca, Graham E. Fagg, Edgar Gabriel, Jack J. Dongarra, "Performance Analysis of MPI Collective Operations". Wokshop on Performance Modeling, Evaluation and Optimisation for Parallel and Distributed Systems (PMEO), in IPDPS 2005.

[QUI03] Martin Quinson, "Découverte automatique des caractéristiques et capacités d'une plateforme de calcul distribué". PhD thesis, École normale supérieure de Lyon, Décembre 2003.

[RAB97] Rolf Rabenseifner, "A new optimized MPI reduce algorithm". High-Performance Computing-Center, University of Stuttgart, Nov. 1997. Disponible sur http ://www.hlrs.de/mpi/myreduce.html

[RAZ02] Danny Raz, Yuval Shavitt, "New models and algorithms for programmable networks". Computer Networks 38, pp. 311-326, 2002.

[REK95] Yakov Rekhter, Tony Li, "A Border Gateway Protocol 4 (BGP-4)". RFC1771, March 1995.

[SAN02] Peter Sanders, Jesper Larsson Träff, "The Hierarchical Factor Algorithm for All-to-All Communication". Euro-Par 2002, LNCS 2400, pp. 799-803, 2002.

[SCH97] André Schiper, "Early consensus in an asynchronous system with a weak failure detector". Distributed Computing, 10(3) Springer-Verlag, Berlin Heidelberg New York, pp. 149-157, 1997.

[SCH02] André Schiper, "Distributed Algorithms : Lecture notes for the Graduate School in Computer Science EPFL-LSR", 2002.

[SEI98] Rich Seifert, "Gigabit Ethernet Technology And Applications For High Speed LANs". Addison-Wesley, 1998.

[SHA99] Gary Shao, Francine Berman, Richard Wolski, "Using Effective Network Views to Promote Distributed Application Performance". In International Conference on Parallel and Distributed Processing Techniques and Applications, June 1999.

[SKI97] David Skillicorn, Jonathan Hill, William McColl, "Questions and answers about BSP". *Scientific Programming*, 6(3), pp. 249-274, 1997.

[SNI96] Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, Jack Dongarra, "MPI : The Complete Reference". The MIT Press, 1996.

[SQU00] Jeffrey Squyres, Andrew Lumsdaine, William George, John Hagedorn, Judith Devaney, "The Interoperable Message Passing Interface (IMPI) Extensions to LAM/MPI". In Proceedings, MPI Developer's Conference, Cornell, NY, USA, 2000.

[STO96] Ion Stoica, Florian Sultan, David Keyes, "A Hyperbolic Model for Communication in Layered Parallel Processing Environnements". *Journal of Parallel and Distributed Computing*, vol. 39, pp. 29-45, 1996.

[SUN02] Yuzhong Sun, David A. Bader, Xiaola Lin, Yibei Ling, "Broadcast on Clusters of SMPs with Optimal Concurrency". International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02), June 2002.

[SUP99] Bronis R. de Supinski, Nicholas T. Karonis, "Accurately Measuring MPI Broadcasts in a Computational Grid". 8th IEEE International Symposium on High Performance Distributed Computing (HPDC'99), 3-6 August, 1999.

[TAM99] Anthony Tam Tat Chun, Cho-Li Wang, "Realistic communication model for parallel computing on cluster". In : *Proc. of the International Workshop on Cluster Computing*, pp. 92-101, 1999.

[TAM01] Anthony Tam Tat Chun, "Performance Studies of High-Speed Communication on Commodity Cluster". PhD Thesis, University of Hong Kong, 2001.

[TAM03] Anthony Tam Tat Chun, Cho-Li Wang, "Contention-Aware Communication Schedule For High-Speed Communication". Cluster Computing : The Journal of Networks, Software Tools and Application, pp. 337-351, Vol. 6, No. 4, 2003.

[TAN02] Wi Bing Tan, Peter Strazdins, "The Analysis and Optimization of Collective Communications on a Beowulf Cluster". Proceedings of the Ninth International Conference on Parallel and Distributed Systems (ICPADS'02), pp. 659-666, 2002.

[THA03] Rajeev Thakur, William Gropp, "Improving the Performance of Collective Operations in MPICH". In Euro PVM/MPI 2003, Springer-Verlag, LNCS Vol. 2840, pp. 257-267, 2003.

[THA05] Rajeev Thakur, Rolf Rabenseifner, William Gropp, " Optimization of Collective Communication Operations in MPICH". International Journal of High Performance Computing Applications, 19 :49-66. 2005.

[TOU99] Juan Touriño, Ramón Doallo, "Modeling MPI Collective Communications on the AP3000 Multicomputer", PVM/MPI'99, LNCS 1697, pp. 133-140, 1999.

[TRA04] Jesper Larsson Träff, "A Simple Work-Optimal Broadcast Algorithm for Message-Passing Parallel Systems". In. Euro PVM/MPI 2004, LNCS 3241, pp. 173-180, 2004.

[URB01] Pascal Urbán, Xavier Défago, André Schiper, "Neko : A single environment to simulate and prototype distributed algorithms". Proceedings of the 15th Int'l Conf. on Information Networking, Beppu City, Japan, 2001.

[VAD00] Sathish Vadhiyar, Graham Fagg, Jack Dongarra, "Automatically Tuned Collective Communications". In : Supercomputing 2000, Dallas TX. IEEE Computer Society, 2000.

[VAD04] Sathish Vadhiyar, Graham Fagg, Jack Dongarra, "Towards an Accurate Model for Collective Communications". International Journal of High Performance Computing Applications, 8(1), pp. 159-167, 2004.

[VAL90] Leslie G. Valiant, "A bridging model for parallel computation". *Communications of the ACM*, 33(8), pp. 103-111, 1990.

[VOR03] Theewara Vorakosit, Putching Uthayopas, "Generating an Efficient Dynamic Multicast Tree under Grid Environnement". In. Euro PVM/MPI 2003, LNCS 2840, pp 636-643, 2003.

[WAG04] Frédéric Wagner, Emmanuel Jeannot, "Message Scheduling for Data Redistribution through High Performance Networks". DistRibUtIon de Données a grande Echelle - DRUIDE'2004, Le Croisic, France, 2004.

[WHE95] Brian Whetten, Simon Kaplan, Todd Montgomery, "A High Performance Totally Ordered Multicast Protocol". In : K. Birman, F. Mattern, A. Schiper (eds.) : Theory and Practice in Distributed Systems : International Workshop. Springer-Verlag, Berlin Heidelberg New York, pp.33-57, 1995.

[WOL97] Rich Wolski, Neil Spring, Chris Peterson, "Implementing a Performance Forecasting System for Metacomputing : The Network Weather Service". In : Supercomputing 1997, 1997.

[WUJ04] Jan-Jan Wu, Shih-Hsen Yeh, Pangfeng Liu, "Efficient Multiple Multicast on Heterogeneous Network of Workstations". The Journal of Supercomputing, 29, pp. 59-88, 2004.

[WUM04] Meng-Shiou Wu, Ricky A. Kendall, Srinivas Aluru, "Exploring Collective Communications on A Cluster of SMPs". Proceedings of the Seventh International Conference on High Performance Computing and Grid in Asia Pacific Region (HPCAsia'04), 2004.

[YAN00] Yuanyuan Yang, Jianchao Wang, "Optimal All-to-All Personalized Exchange in Multistage Networks". International Conference on Parallel and Distributed Systems (ICPADS'00), pp. 229-236, 2000.

Résumé

Avec la démocratisation des environnements du type grappe et grille de calcul, la performance des opérations de communication collective devient un aspect critique dans le coût total des applications parallèles. Fortement influencées par l'hétérogénéité des ressources, ces opérations dépendent à la fois des paramètres de communication des réseaux et des stratégies de communication employées.

Cette thèse a pour objet d'étude l'optimisation des communications collectives selon l'approche préconisée par Karonis, où les différentes grappes de calcul sont organisées en plusieurs couches de communication de manière à minimiser le temps total de communication. Pour cela, nous proposons des modèles de communications qui permettent l'optimisation des communications collectives à travers l'ordonnancement hiérarchique des communications et la prédiction des performances.

Dans un premier temps, nous démontrons que l'utilisation du modèle de coût pLogP permet la modélisation des performances de communications collectives dans des environnements homogènes. Ces modèles de performance ont été validés expérimentalement sur différentes plates-formes réseaux (Fast Ethernet, Giga Ethernet et Myrinet).

Parallèlement, nous étudions la découverte automatique de la topologie du réseau. En effet, la décomposition de l'environnement de grille en îlots d'homogénéité permettrait la réduction de la complexité des optimisations, notamment dans ce qui concerne la modélisation des performances et l'obtention des caractéristiques du réseau.

Notre principale contribution a été de proposer certaines heuristiques d'ordonnancement des communications qui tiennent compte de l'organisation hiérarchique de la grille. Ces heuristiques, adaptées aux différents patrons de communication collective, utilisent les techniques étudiées précédemment (dont la découverte des îlots d'homogénéité et la modélisation des performances) afin de réduire la complexité de l'optimisation des communications et de minimiser le temps total de communication.

Mots clés : communication collective, modélisation de performance, grilles de calcul, MPI

Abstract

The popularity of heterogeneous parallel processing environments like clusters and computer grids has emphasised the impact of network heterogeneity on the performance of parallel applications. Collective communication operations are especially concerned by this problem, as communication heterogeneity interfers directly on the performance of the communication strategies. Therefore, the aim of this dissertation is the study the optimisation of collective communications in heterogeneous systems like computational grids.

We investigate how to optimise communications according to the approach proposed by Karonis, where the communication among the different clusters that compose a grid are organised in multiple levels, minimising the total communication time (makespan). Hence, in a first moment, we study how to accurately model the performance of collective communications in homogeneous clusters, and, we validate such models under different network architectures. In parallel, we study how to minimise the impact of network heterogeneity on the complexity of communication scheduling techniques. Therefore, we investigated how topology discovery techniques can be used to decompose the grid environment into islands of homogeneous clusters, more adapted to an hierarchical modelling of the network.

Finally, we study how performance modelling and topology discovery can be associated to construct grid-aware collective communications. Using the information from both performance models and network topology, we compare different communication scheduling heuristics present on the literature. We also propose new heuristics to schedule inter-cluster communications, taking into account the performance of each different cluster. These heuristics seem to be specially efficient when the number of interconnected clusters augments, a tendency for the next years.

Keywords: collective communication, performance modelling, metacomputing, MPI