Architecture d'une plate-forme de communication multi-paradigme pour les grilles

Alexandre Denis

RENPAR'15 / CFSE'3 / SympAAA'2003
La Colle sur Loup, France, 15 au 17 octobre 2003

Architecture d'une plate-forme de communication multi-paradigme pour les grilles *

Alexandre Denis

IRISA/IFSIC,
Campus de Beaulieu,
35042 RENNES Cedex — France
Alexandre.Denis@irisa.fr

Résumé
Cet article étudie un modèle de plate-forme de communications dont le but est d'étendre la portée des grilles de calcul en permettant l'exécution d'applications parallèles et/ou réparties sans imposer de contrainte de programmation ou d'exécutif particulier. Notre objectif n'est pas de montrer qu'un exécutif est meilleur qu'un autre, mais plutôt de construire une plate-forme de communications pour les grilles qui permet l'utilisation des exécutifs adaptés à l'application plutôt que ceux dictés par les réseaux disponibles. Une telle plate-forme doit ainsi être capable de fournir n'importe quel exécutif — éventuellement plusieurs en même temps — sur n'importe quel réseau.

Mots-clés : Grille de calcul, intergiciel, réseau, parallélisme, calcul réparti

1. Introduction

L'émergence récente des grilles de calcul donne accès à des ressources de calcul à une échelle sans précédent dans l'histoire du calcul scientifique. La question cruciale qui se pose alors est celle de la méthode et du modèle de programmation à employer pour les programmer. En effet, les grilles sont constituées de ressources de calcul variées, distribuées à grande échelle ; ce sont par exemple des supercalculateurs, des grappes de PC, interconnectés par des technologies réseaux variées — des SAN à l'intérieur d'une pièce jusqu'au WAN à l'échelle d'un continent. Une façon courante de programmer une telle infrastructure de calcul est de lui donner l'apparence d'un calculateur *parallèle* virtuel. Les programmeurs utilisent alors les techniques habituelles de programmation parallèle, par exemple en se basant sur MPI. Cependant, une telle approche risque de limiter arbitrairement la portée des grilles uniquement aux applications parallèles. Des applications basées sur des exécutifs du *calcul réparti* tels que Corba [12], HLA [10], Java ou Soap [5] sont également de bons candidats pour les grilles. Enfin, certains modèles de programmation tels que les objets ou les composants parallèles [14] demandent *une combinaison* d'exécutifs du parallélisme et du calcul réparti et sont également appropriés à un déploiement sur les grilles.
Par conséquent, un environnement logiciel pour la programmation des grilles se doit de supporter les différents modèles de programmation et d'exécution. Il ne doit pas contraindre le programmeur à utiliser uniquement MPI comme c'est le cas avec MPICH-G2 [9]. Cet article présente un modèle de plate-forme de communications pour les grilles de calcul qui donne accès à un vaste choix d'exécutifs différents et leur permet d'utiliser efficacement tous les réseaux disponibles sur les grilles.
La section 2 présente une analyse du modèle de communication utilisé sur les grilles. La section 3 propose un modèle de plate-forme de communication qui supporte à la fois le parallélisme et le réparti. La section 4 présente une implémentation de ce modèle au sein de la plate-forme PadicoTM et évalue ses résultats. La section 5 décrit quelques travaux connexes. Enfin nous donnons notre conclusion en section 6.

* Ces travaux sont supportés par l'ACI GRID "RMI" du ministère de la recherche.

2. Analyse du modèle de communication sur les grilles

2.1. Caractéristiques de l'usage du réseau sur les grilles

Pour une application donnée, le déploiement sur une grille peut mener à des configurations différentes. Par exemple, un schéma de déploiement peut être un ensemble de nœuds sur une même grappe de PC équipée d'un réseau haute performance, un autre schéma peut mettre en jeu deux grappes reliées par un WAN. Les applications peuvent être écrites en utilisant des composants parallèles [1, 14] où un composant logiciel contient un code parallèle ; dans ce cas, la plate-forme de composants utilise son propre mécanisme de communications qui est distinct de celui utilisé à l'intérieur des composants pour le parallélisme. Par exemple, un composant parallèle utilisant MPI en interne peut être connecté à un composant qui utilise PVM, la connexion entre les composants étant réalisée par Corba. De cet exemple, nous déduisons les caractéristiques qu'une plate-forme de communications pour les grilles doit présenter :

Transparence — Les divers exécutifs utilisés par une application doivent être capables d'utiliser les ressources réseau disponibles de façon transparente, efficace et automatique. Aussi bien les communications MPI, PVM, Corba, Soap que Java doivent être capables de tirer profit d'un réseau haute performance s'il est présent ou de s'adapter à un WAN si nécessaire.

Flexibilité — Les applications utilisent une large palette d'exécutifs différents, et nous pouvons légitimement supposer que ceci restera vrai à l'avenir. Il semble plus judicieux de faciliter la "gridification" des exécutifs plutôt que de lier les applications à une plate-forme et/ou exécutif particulier.

Interopérabilité — Les grilles ne sont pas des mondes fermés. Les applications doivent être accessibles *via* des protocoles standard.

Offrir plusieurs paradigmes de communication — Certains modèles de programmation tels que les objets et composants parallèles (CCA [1], GridCCM [14], Paco++ [7]) nécessitent plusieurs exécutifs simultanément. C'est le cas également par exemple pour des systèmes de *monitoring* d'applications parallèles par Soap ou Corba par exemple. Il est donc primordial d'être capable d'offrir plusieurs exécutifs simultanément.

2.2. Analyse des paradigmes de communication

Au sens large, un *paradigme* est un modèle auquel on apparente une famille d'exécutifs ; la distinction se fait selon certains critères fixés. Dans cet article, nous faisons la distinction entre le paradigme *parallèle* et le paradigme *réparti* :

Parallèle — En parallélisme, la priorité est la haute performance. Les communications ont lieu à l'intérieur d'un ensemble de nœuds défini et connu de tous, les messages ont des frontières définies, l'API est conçue pour favoriser les implémentation "zéro-copie", des opérations collectives sont fournies et optimisées. L'exemple typique est MPI. Il est courant de faire la distinction entre parallélisme à mémoire distribuée et à mémoire partagée. Dans cet article centré sur le réseau, nous nous intéressons surtout au parallélisme à mémoire distribué.

Réparti — La priorité en réparti est l'interopérabilité. Les connexions sont dynamiques, gérées lien par lien en client/serveur ; l'interopérabilité est assurée entre les architectures, systèmes d'exploitation et éditeurs de logiciels différents ; les échanges de données peuvent utiliser des flux continus. Les exemples typiques sont TCP/IP, Soap ou Corba.

Ces définitions ne constituents pas une classification stricte. Il faut plutôt les comprendre comme des tendances pour appréhender facilement les caractéristiques principales du modèle de communication des exécutifs. Dans la suite de cet article, nous considérons que TCP/IP, Corba, Soap, HLA et Java RMI sont répartis ; MPI, PVM, DSM, *FastMessage*, Madeleine ou *Panda* sont parallèles.

2.3. Analyse des niveaux d'abstraction des ressources

L'abstraction consiste en la définition d'une interface dite *abstraite* qui existe indépendamment de toute implémentation particulière. Il peut exister plusieurs incarnations qui implémentent la même interface abstraite. L'abstraction est un mécanisme largement utilisé pour s'affranchir des différences entre les différents types de réseaux. Dans ce cas, il s'agit de *portabilité*. Quand une interface abstraite est conçue avec pour objectif d'être implémentée au-dessus de plusieurs réseaux mais aussi d'être utilisable par

plusieurs applications ou exécutifs, alors nous la qualifions de *générique*. L'empilement des interfaces constitue des couches dont le niveau d'abstraction augmente de bas en haut :

Niveau système — implémenté par un *pilote*, tel que GM, BIP, Via ou Sisci ou par le système d'exploitation pour les *sockets* TCP/IP. Le niveau d'abstraction est faible : un *pilote* différent est utilisé pour chaque type de matériel.

Niveau générique — implémenté par une plate-forme générique de communication, telle que Madeleine, *Panda* ou *Nexus*. L'interface proposée est uniforme d'un réseau à l'autre.

Niveau applicatif — implémenté par un exécutif, tel que MPI, Corba ou HLA. L'interface proposée est prévue pour être utilisée directement par les applications.

3. Un modèle de plate-forme d'intégration d'exécutifs

Dans cette section, nous proposons un modèle de plate-forme de communications pour les grilles de calcul qui prenne en compte les exécutifs parallèles et répartis.

3.1. Intégrer les abstractions

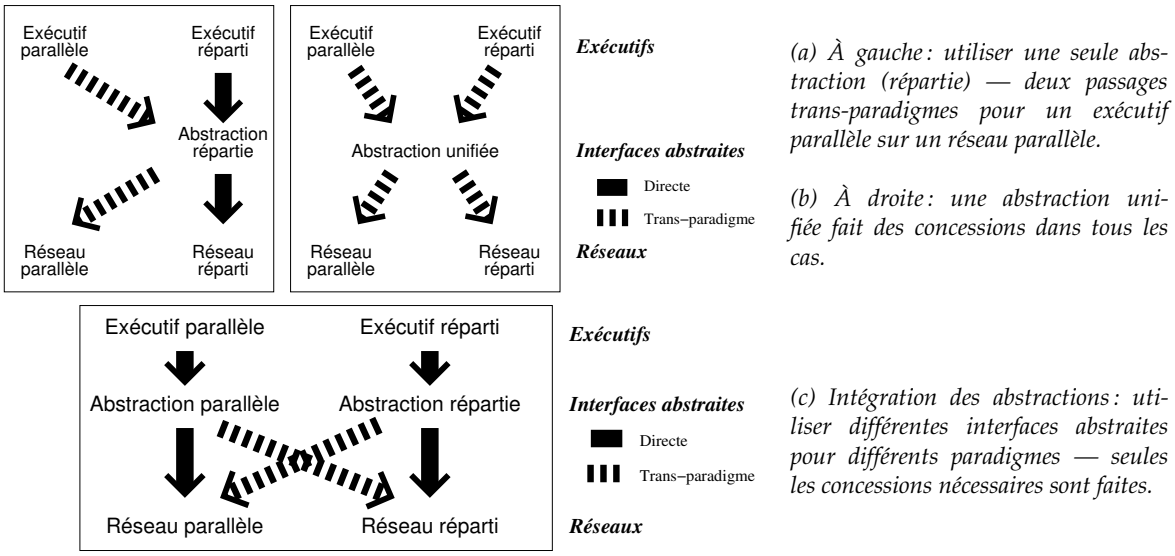
Le modèle d'abstraction "classique" apporte la *portabilité*, c'est-à-dire la capacité pour un exécutif d'utiliser plusieurs types de réseaux différents en fonction de ce qu'il a à sa disposition, et la *généricité*, c'est-à-dire la possibilité de réutiliser les infrastructures logicielles de portabilité d'un exécutif à l'autre. Cependant, la généricité telle qu'elle est mise en œuvre habituellement est basée sur la définition d'une interface abstraite unique. Ce choix est particulièrement pertinent du point de vue de la portabilité, mais peut sembler discutable pour la généricité : la généricité reste à l'intérieur d'un paradigme particulier.

Pourtant, bon nombre d'exécutifs parallèles peuvent utiliser des *sockets* TCP/IP qui sont une abstraction répartie. Cette approche est valide lorsqu'il s'agit d'utiliser un réseau orienté réparti (Ethernet, par exemple), mais elle semble beaucoup moins pertinente pour utiliser un réseau conçu pour le parallélisme (réseau interne d'un supercalculateur ou d'une grappe, par exemple). Comme l'illustre la figure 1 (a), l'utilisation exclusive d'une abstraction répartie impose les compromis du réparti même aux exécutifs parallèles. Par exemple une implémentation MPI qui serait bâtie sur TCP/IP pourrait certes utiliser le réseau interne d'une machine parallèle par TCP/IP, mais ne pourrait pas tirer profit des propriétés "parallèles" de ce réseau (opérations collectives optimisées par exemple) car ces propriétés ne sont pas exprimables par l'interface de TCP/IP. Symétriquement, il est courant d'utiliser une unique abstraction parallèle comme dénominateur commun sur les grilles, par exemple à MPICH-G2 [9]. Il est possible de l'utiliser pour implémenter des mécanismes de communications de type réparti tels que des objets distribués. Cependant, l'interface MPI orientée vers le parallélisme ne permet pas d'exprimer des propriétés essentielles du réparti telles que l'adressage IP, la dynamicité avec connexions client/serveur (au lieu du *spawn*), ou encore l'interopérabilité avec d'autres implémentations. En particulier il ne semble pas possible de réaliser une implémentation Corba complète conforme au standard au-dessus de MPICH-G2.

Dans les deux cas, l'utilisation d'une interface abstraite orientée vers le parallélisme ou vers le réparti pénalise les exécutifs de l'autre type car certaines propriétés fournies par le niveau système ne sont pas exprimables au niveau abstrait donc sont perdues.

Au regard de ces conclusions, nous pourrions être tentés d'essayer de définir une abstraction unique qui combinerait les propriétés du parallélisme et du réparti, comme représenté à la figure 1 (b). Pour tenir compte des contraintes d'interopérabilité du réparti, nous remarquons qu'une abstraction unifiée ne peut pas être très éloignée du réparti. Il ne semble pas réaliste d'affaiblir les contraintes du réparti pour se placer dans les hypothèse plus relâchées qui permettent les optimisations en parallélisme : changer l'interface de *streaming* pour une interface d'échange de messages casse l'interopérabilité avec TCP/IP ; propager des informations de topologie et de configuration matérielle pour optimiser les opérations collectives semble incompatible avec les connexions gérées lien par lien dynamiquement et l'interopérabilité avec le protocole IP standard du réparti. Une abstraction unifiée revient donc à imposer inutilement les compromis du réparti au parallélisme qui n'en a pas besoin. Le choix d'une seule abstraction de niveau générique, qu'elle soit parallèle, répartie, ou unifiée, ne semble donc pas satisfaisant.

Plutôt que de chercher à accorder les contraires, nous proposons de juxtaposer une interface abstraite parallèle et une autre répartie. En effet, chaque exécutif est soit parallèle, soit réparti, mais pas les deux



(a) À gauche : utiliser une seule abstraction (répartie) — deux passages trans-paradigmes pour un exécutif parallèle sur un réseau parallèle.

(b) À droite : une abstraction unifiée fait des concessions dans tous les cas.

(c) Intégration des abstractions : utiliser différentes interfaces abstraites pour différents paradigmes — seules les concessions nécessaires sont faites.

FIG. 1 – *Différents modèles d'abstraction envisageables*

en même temps. Par exemple Corba, HLA et Soap ont besoin d'une abstraction répartie, alors que MPI ou PVM ont besoin d'une abstraction parallèle. Il suffit donc de fournir chacune des deux abstractions sur les deux types de réseau. Les compromis qui permettent d'exprimer les deux paradigmes en une seule abstraction sont superflus. Cette approche est illustré à la figure 1 (c). Chaque exécutif utilise l'abstraction qui lui convient. Chaque abstraction est instanciée sur chaque réseau par un *adaptateur* ; les adaptateurs sont soit *directs*, soit *trans-paradigme*. De cette façon, les compromis pour passer d'un paradigme à l'autre ne sont fait que lorsqu'ils sont nécessaires, c'est-à-dire pour un exécutif et un réseau de paradigme différent. De ce fait, les propriétés d'un réseau utilisé par un exécutif de même paradigme sont toujours exprimables au niveau générique. Nous choisissons de bâtir notre plate-forme de communications pour les grilles suivant ce modèle d'abstractions.

3.2. Virtualiser les ressources pour un changement transparent de méthode de communication

Les exécutifs susceptibles d'être utilisés sur une grille sont variés : MPI, Corba, Soap, HLA, PVM, JVM, etc. De plus, pour chaque type d'exécutif, il existe plusieurs implémentations, chacune ayant ses spécificités. Le développement d'un exécutif est une tâche lourde — à titre d'exemple, MPICH contient 200 000 lignes de code — et qui demande des compétences très pointues. De plus, les normes — et donc les exécutifs — évoluent sans cesse. Il ne nous semble donc pas raisonnable de redévelopper une implémentation de chacun des exécutifs spécifiquement pour une plate-forme de communications. Nous choisissons plutôt de réutiliser les implémentations existantes de façon à suivre automatiquement leurs évolutions et à pouvoir profiter de leurs spécificités.

La méthode employée pour utiliser de façon transparente un exécutif existant consiste en la *virtualisation* des ressources. Il s'agit de présenter à l'exécutif l'API qu'il attend, même si en fait les ressources utilisées sont différentes. Par exemple, on présente une interface standard *socket* à une implémentation Corba qui "croit" ainsi utiliser TCP/IP. Les API présentées aux exécutifs sont implémentées par de fines couches logicielles, appelées *personnalités*, au-dessus des abstractions pour leur donner l'aspect voulu. Il est possible de fournir plusieurs personnalités en même temps au-dessus d'une abstraction selon les besoins des exécutifs.

Les mécanismes de virtualisation et d'abstraction avec adaptateurs trans-paradigme permet l'utilisation de n'importe quel exécutif sur n'importe quel type de réseau. Cependant, quand l'utilisation d'un adaptateur *direct* est possible, ce n'est pas forcémment la meilleure solution. Il est parfois souhaitable d'utiliser une autre méthode, particulièrement en réparti. Ces méthodes sont par exemple :

Flux parallèles sur WAN — Sur les WAN à forte latence, la moindre perte de paquet IP peut être drama-

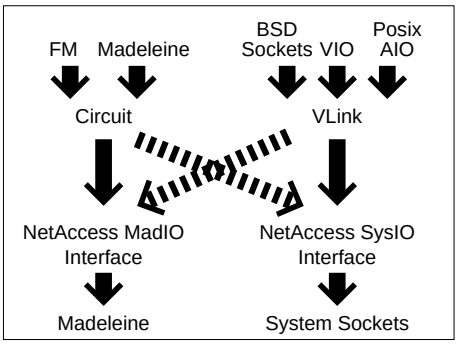


FIG. 2 – *Implémentation du modèle de communication dans PadicoTM.*

tique pour les performances avec TCP/IP. Le coût est amorti en répartissant les communications sur plusieurs flux parallèles, comme le fait *GridFTP*.

Compression à la volée — Sur les réseaux à très faible débit, le débit utile peut être amélioré en compressant les données à la volée comme le fait *AdOC*.

Chiffrement et authentification — Pour les connexions entre deux sites, il est plus prudent d'utiliser un mécanisme d'authentification des connexions. Il est utile de chiffrer les communications pour les données sensibles.

Ces différentes méthodes s'inscrivent parfaitement dans le modèle proposé en étant fournies sous formes d'*adaptateurs alternatifs* qui incarnent une autre implémentation de l'interface abstraite. Leur utilisation est donc transparente du point de vue de l'exécutif.

3.3. Un modèle hybride parallèle + réparti

Nous proposons un modèle de plate-forme de communications basé sur le modèle d'abstraction et de virtualisation décrits ci-dessus. Ce modèle est organisé en trois couches — arbitrage, abstraction, personnalités (de bas en haut). Son implémentation est représentée à la figure 2.

Arbitrage — L'accès concurrent au réseau par plusieurs exécutifs en même temps est une grande source de conflits potentiels. Nous proposons d'arbitrer l'accès aux ressources au plus bas niveau de façon à pouvoir construire par dessus une plate-forme entièrement réentrante.

Abstraction — La couche d'abstraction proposes deux abstractions, parallèle et répartie. Chaque abstraction est implémentée en diverses variantes par des *adaptateurs* : directs, trans-paradigmes, et alternatifs. L'interface ne varie pas selon l'adaptateur utilisé.

Personnalités — Les personnalités adaptent l'API générale des abstractions pour la transformer en l'API attendue par les exécutifs. Ceci permet la réutilisation transparente d'une grande variété d'exécutifs adaptés aux besoins des applications.

4. Implémentation du modèle dans PadicoTM

Le modèle décrit à la section précédente a été implémenté dans PadicoTM, notre plate-forme de recherche pour explorer le domaine de l'intégration de plusieurs exécutifs communicants. Cette plate-forme est utilisée dans le cadre des objets Corba parallèles [7] qui ont besoin à la fois de Corba et de MPI. PadicoTM résout des problèmes tels que la cohabitation d'exécutifs, le chargement dynamique de code et l'utilisation efficace du *multi-threading*. Ces points ne font pas l'objet de cet article centré sur les communications et sont décrits dans [8]. L'implémentation du modèle est illustrée par la figure 2.

4.1. Arbitrage : *NetAccess*

NetAccess est la couche d'arbitrage d'accès aux réseaux. *NetAccess* est subdivisé en deux modules : *SysIO* qui gère l'accès collaboratif aux entrées/sorties systèmes (*sockets*, fichiers) et *MadIO* qui fournit un mul-

tiplexage au-dessus de Madeleine pour l'accès aux réseaux haute performance.

NetAccess MadIO — Pour obtenir de bonnes performances et une bonne portabilité sur les réseaux haute performance, nous avons choisi d'utiliser la bibliothèque Madeleine [3] comme fondation. Madeleine fournissant un nombre de canaux de communications limité aux capacités du matériel, *MadIO* ajoute des mécanismes de multiplexage qui permettent un nombre arbitraire de canaux. Les en-têtes ajoutés aux messages par *MadIO* pour le multiplexage sont agrégés avec ceux des autres couches de façon à ne pas pénaliser la latence. Nous avons mesuré que le surcoût effectif du multiplexage est de l'ordre de 0.1 μs ce qui est négligeable sur la plupart des réseaux.

NetAccess SysIO — Contrairement à une croyance répandue, l'API *socket* n'apporte pas réentrance, multiplexage et coopération. Il existe des risques de conflits avec des entrées/sorties qui utilisent les signaux Unix, et des problèmes d'inéquité voire de famine en cas de juxtaposition d'un exécutif qui fait de l'attente active et d'un autre en attente bloquante. *SysIO* résout ces problèmes en factorisant la gestion des *sockets* systèmes dans une boucle gérée par PadicoTM. La réentrance est assurée par l'utilisation de *callbacks* enregistrés par l'utilisateur, et appelés par PadicoTM quand l'accès au réseau est jugé opportun.

***NetAccess* core** — Le cœur de *NetAccess* gère l'entrelacement entre *MadIO* et *SysIO*. Selon les besoins, la priorité peut être donnée à l'un ou l'autre par une API de configuration accessible à l'utilisateur.

4.2. Abstractions : *VLink* et *Circuit*

Les deux abstractions du modèle de communication dans PadicoTM sont *VLink Circuit* :

Interface pour le réparti : *VLink* — L'interface *VLink*, conçue pour le réparti, est orientée client-serveur, supporte les connexions/déconnexions dynamiques et le *streaming*. Pour ne pas brider le choix de personnalités, *VLink* présente une API asynchrone : les opérations sont invoquées de manière non-bloquante, et leur état d'avancement peut être consulté dans le descripteur de connexion. Si besoin, il est possible d'attendre de manière synchrone la terminaison de l'opération invoquée. Ces opérations sont : `read`, `write`, `connect`, `accept`, `close`. Un tel ensemble de fonctions constitue un *pilote VLink* . Des pilotes *VLink* ont été implémentés sur *SysIO*, *MadIO*, sur des flux parallèles pour WAN et en *loopback*.

Interface pour le parallélisme : *Circuit* — L'interface *Circuit*, conçue pour le parallélisme, est orientée SPMD, sur un *groupe* de nœud défini, permet les communications de tous les nœuds à tous les autres et utilise une API d'échange de messages délimités optimisée inspirée de celle de Madeleine. Un *groupe* de nœuds est un ensemble qui peut être arbitraire en répartition géographique, à l'intérieur d'une grappe, à cheval sur plusieurs grappes voire plusieurs sites. Les implémentations de *Circuit* ont été réalisées sur *MadIO*, *SysIO*, *loopback* intra-processus, et *VLink*. Un *Circuit* peut utiliser plusieurs pilotes en même temps selon les nœuds du groupe. Les travaux sur l'intégration de opérations collectives dans *Circuit* sont en cours.

Choix de protocole — Le choix de l'adaptateur à utiliser par *VLink* ou *Circuit* est automatique, en fonction d'une base de connaissance de la topologie gérée par PadicoTM, et de préférences données par l'utilisateur. Les adaptateurs alternatifs étant de type réparti, l'adaptateur *Circuit/VLink* permet d'utiliser ces adaptateurs alternatifs aussi bien avec *VLink* que *Circuit*.

4.3. Personalités et exécutifs supportés

Au-dessus des deux abstractions, PadicoTM fournit des personnalités qui présentent des API standard de façon à pouvoir utiliser de façon transparente des exécutifs ou des applications déjà écrits. Les personnalités proposées sur *VLink* sont : VIO pour une utilisation explicite par une API de type *sockets*, *SysWrap* pour utiliser directement du code C, C++ ou FORTRAN prévu pour *sockets* BSD standard sans même avoir besoin de le recompiler, et AIO qui est une implémentation de l'API *Posix.2 Asynchronous Input/Output Interface*. Les personnalités proposées au-dessus de *Circuit* sont *FastMessage* 2.0 et Madeleine. Grâce à la personnalité *SysWrap*, le portage de bon nombre d'exécutifs répartis est quasi-immédiat, sans modifier le code source. Nous avons ainsi testé les implémentations Corba omniORB 3, omniORB 4, ORBacus 4.0, Mico 2.3, l'implémentation HLA *Certi* 3.0, et l'implémentation Soap *gSoap* 2.2. Le portage de la machine virtuelle Java *Kaffe* 1.0.7 a nécessité quelques ajustements au niveau du *multi-threading* et de la gestion des signaux. Grâce à la personnalité Madeleine de *Circuit*, l'implémentation existante MPICH/Madeleine [2] peut être utilisé dans PadicoTM. Enfin, les mécanismes de chargement dyna-

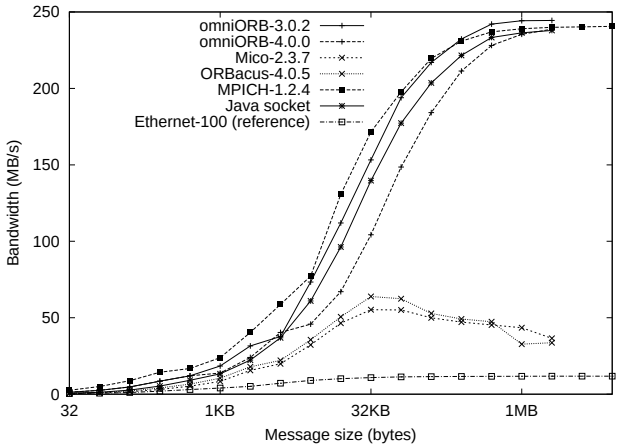


FIG. 3 – *Débit obtenu par différents exécutifs avec PadicoTM sur Myrinet-2000.*

mique et d'arbitrage autorisent n'importe quelle combinaisons d'exécutifs chargés en même temps et leur permet d'emblée d'utiliser efficacement tous les réseaux supportés par PadicoTM.

4.4. Évaluation des performances

Notre plate-forme de tests est composée de bi-Pentium II 1 GHz équipés de 512 Mo de mémoire et de cartes Ethernet-100 et Myrinet-2000. Le débit obtenu par les différents exécutifs sur Myrinet-2000 est représenté à la figure 3. Nous observons que presque tous les exécutifs — MPICH, omniORB, Java — parviennent à exploiter la bande passante disponible à hauteur de 240 Mo/s, soit 96 % du maximum matériel. ORBacus et Mico, deux implémentations Corba qui, à la différence d'omniORB, effectuent des copies en mémoire, obtiennent repsectivement 63 et 55 Mo/s ; ces performance médiocres sont dues à leur implémentation interne et restent conformes avec la théorie [6]. La latence obtenue va de 11 μs pour MPI jusque 20 μs pour omniORB, ce qui est un bon résultat. À notre connaissance, omniORB/PadicoTM est l'implémentation Corba existante la plus rapide. Sur le réseau VTHD entre Rennes et Nice, tous les exécutifs réalisent sensiblement la même performance, à savoir une latence de 8 ms et un débit de l'ordre de 9 Mo/s, ce qui est la performance typique que l'on est en droit d'attendre d'un tel réseau. L'activation des flux parallèles permet d'augmenter le débit jusque 12 Mo/s ce qui est le maximum possible compte tenu du fait que les machines sont connectées à VTHD par Ethernet-100.

5. Travaux connexes

Il existe à notre connaissance très peu de travaux dans le domaine de l'intégration d'exécutifs de plusieurs paradigmes, la plupart des travaux concernant un exécutif ou un paradigme en particulier. Par exemple Panda [4] est un environnement générique dédié aux exécutifs parallèles, notamment MPI et PVM. ADAPTIVE [15] (ACE) est un environnement générique pour le réparti. Ni l'un ni l'autre n'est conçu pour servir de base à des exécutifs parallèles *et* répartis. *Harness* [11] et *Quarterware* [16] permettent l'utilisation de plusieurs exécutifs mais sont restreints respectivement à MPI+PVM et MPI+RMI, et sont limités à TCP/IP. En ce qui concerne les réseaux haute performance, leur utilisation est courante avec MPI. Elle l'est beaucoup moins en réparti, et avec Corba en particulier. Quelques travaux [13] ont mené à des prototypes Corba sur ATM ou SCI ; suites aux résultats décevants, ces pistes ont été abandonnées.

6. Conclusion

Il est probable que les applications des grilles de calcul utilisent des exécutifs variés. Ces exécutifs devront offrir transparence, flexibilité et efficacité. Cet article a introduit un modèle de communications pour les grilles de calcul basé sur un carrefour entre les mondes du parallélisme et du calcul réparti : les

deux paradigmes sont présents au niveau des infrastructures supportées et des exécutifs proposés. La force de ce modèle est sa flexibilité et sa transparence obtenues sans dégradation de performance. Sa capacité à supporter plusieurs exécutifs simultanément en fait une plate-forme de choix pour les modèles à objets parallèles. Nous avons présenté les aspects liés au réseau dans la plate-forme PadicoTM qui réalise une implémentation de ce modèle. Cette plate-forme supporte de nombreuses méthodes d'accès aux réseaux : BIP, GM, Sisci, Via, TCP/IP, flux parallèles, *AdOC*, et par l'intermédiaire des personnalités adaptées supporte des exécutifs variés : MPI, plusieurs implémentations Corba, HLA, Soap, Java.
Les aspects liés à la sécurité (chiffrement, authentification) feront l'objet de travaux futurs en utilisant des mécanismes existants tels que GSI ou IPSEC. Les autres travaux futurs concernent d'autres méthodes de communications (tunnels pour traverser les *firewalls*, adressage global en dehors du système IP) et d'autres exécutifs tels qu'une DSM. PadicoTM est un logiciel libre distribué sous license *Gnu*. Il peut être téléchargé à l'adresse `http://www.irisa.fr/paris/Padicotm/`.

Bibliographie

1. Armstrong (R.), Gannon (D.), Geist (A.), Keahey (K.), Kohn (S.), McInnes (L.), Parker (S.) et Smolinski (B.). – Toward a common component architecture for high-performance scientific computing. *In : Proceeding of the 8th IEEE International Symposium on High Performance Distributed Computation*.
2. Aumage (O.), Mercier (G.) et Namyst (R.). – MPICH/Madeleine: a true multi-protocol MPI for high-performance networks. *In : Proc. 15th International Parallel and Distributed Processing Symposium (IPDPS 2001)*. IEEE, p. 51. – San Francisco, avril 2001.
3. Aumage (Olivier), Bougé (Luc), Denis (Alexandre), Méhaut (Jean-François), Mercier (Guillaume), Namyst (Raymond) et Prylli (Loïc). – A portable and efficient communication library for high-performance cluster computing. *In : IEEE Intl Conf. on Cluster Computing (CLUSTER 2000)*, pp. 78–87. – Technische Universität Chemnitz, Saxony, Germany, novembre 2000.
4. Bal (Henry), Benson (Gregory), Bhoedjang (Raoul), Langendoen (Koen) et Rühl (Tim). – Experience with a portability layer for implementing parallel programming systems. *In : International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'96)*, pp. 1477–1488. – Sunnyvale, CA, août 1996.
5. Cerami (Ethan). – *Web Services Essentials*, chap. Simple Object Access Protocol (SOAP), pp. 49–112. – O'Reilly & Associates, février 2002, 1st édition.
6. Denis (A.). – CORBA haute performance. *Technique et Science Informatiques (TSI)*, vol. 21, 2002, pp. 659–683.
7. Denis (A.), Pérez (C.) et Priol (T.). – Portable parallel CORBA objects: an approach to combine parallel and distributed programming for grid computing. *In : Proc. of the 7th Intl. Euro-Par'01 conf.* pp. 835–844. – Manchester, UK, août 2001.
8. Denis (A.), Pérez (C.) et Priol (T.). – Towards high performance CORBA and MPI middlewares for grid computing. *In : Proc of the 2nd International Workshop on Grid Computing*, éd. par Lee (Graig A.). pp. 14–25. – Denver, Colorado, USA, novembre 2001. In conjunction with *SuperComputing 2001 (SC'01)*.
9. Foster (I.), Geisler (J.), Gropp (W.), Karonis (N.), Lusk (E.), Thiruvathukal (G.), et Tuecke (S.). – Wide-area implementation of the message passing interface. *Parallel Computing*, vol. 24, n12, 1998, pp. 1735–1749.
10. IEEE. – IEEE standard for modeling and simulation (M&S) high level architecture (HLA)—federate interface specification. – IEEE Standard 1516, septembre 2000.
11. Kurzyniec (Dawid), Sunderam (Vaidy) et Migliardi (Mauro). – On the viability of component frameworks for high performance distributed computing: A case study. *In : IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*. – Edimburg, Scotland, juillet 2002.
12. OMG. – The Common Object Request Broker: Architecture and Specification V3.0. – OMG Document formal/02-06-33, juin 2002.
13. Pope (Steve) et Lo (Sai-Lai). – *The Implementation of a Native ATM Transport for a High Performance ORB*. – rapport de recherche, Cambridge, Olivetti & Oracle Laboratory, juin 1998.
14. Pérez (Christian), Priol (Thierry) et Ribes (André). – A parallel CORBA component model for numerical code coupling. *In : Proc. of the 3nd International Workshop on Grid Computing*, éd. par Lee (Craig A.). – Baltimore, Maryland, USA, novembre 2002. to appear.
15. Schmidt (Douglas C.). – An architectural overview of the ACE framework: A case-study of successful cross-platform systems software reuse. *USENIX login magazine, Tools special issue*, novembre 1998.
16. Singhai (Ashish). – *Quarterware: a Middleware Toolkit of Software RISC Components*. – Thèse de PhD, University of Illinois at Urbana-Champaign, 1999.