Adaptation de l'environnement générique de metacomputing GLOBUS à des réseaux haut débit

Alexandre DENIS*

Résumé

L'environnement de metacomputing GLOBUS permet l'utilisation simultanée d'un grand nombre de machines réparties partout dans le monde. Il est malheureusement conçu dans le but de relier des supercalculateurs, et les grappes de PC n'ont pas été prises en compte. Il n'est pas possible d'exploiter les protocoles haute performance tels que BIP ou SCI dans GLOBUS. Notre approche est d'ajouter à GLOBUS un support pour les réseaux rapides répandus dans les grappes en y intégrant la bibliothèque de communication MADELEINE. L'intégration est réalisée de façon à ce que toute application GLOBUS bénéficie de l'amélioration apportée par la présence de MADELEINE sans avoir besoin de la modifier.

Mots-clé : grappe de PC, réseau haut débit, metacomputing, Globus, Madeleine

1 Introduction : metacomputing sur grappes

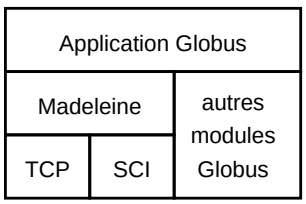
GLOBUS/NEXUS : un environnement de metacomputing Le développement du Web a révolutionné la façon dont nous pensons l'accès à l'information. Il est aujourd'hui possible d'accéder à virtuellement n'importe quelle information électronique dans le monde via le Web. Le metacomputing propose une révolution similaire pour les moyens de calcul et de stockage. Il introduit la notion de grille de calcul ; un des ouvrages fondateurs du metacomputing [5] décrit cette grille comme un ensemble de supercalculateurs dispersés partout dans le monde, reliés par Internet, et disposant d'énormes capacités de stockage. GLOBUS [2] est un tel environnement de metacomputing qui permet la mise en œuvre d'une grille de calcul.

Pour cela, GLOBUS intègre entre autres une puissante bibliothèque de communications appelée NEXUS [4]. NEXUS supporte l'utilisation simultanée de plusieurs protocoles comme par exemple TCP, UDP ou MPI. La sélection du protocole est automatique et entièrement transparente pour l'application. L'utilisation en milieu hétérogène est rendue possible par une conversion automatique des différents formats de données. Enfin, dans le metacomputing, la portabilité est prioritaire sur la performance.

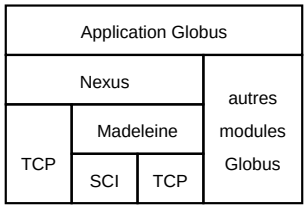
GLOBUS sur les grappes GLOBUS est actuellement inadapté à une utilisation sur des grappes. Il a été conçu comme un environnement de metacomputing destiné à relier de grosses machines parallèles partout dans le monde. Les protocoles de communication supportés par NEXUS sont donc naturellement des protocoles longue distance et les protocoles des réseaux internes de ces machines. Pour la longue distance, TCP prend en charge les connexions fiables, c'est-à-dire la plupart des communication inter-sites. Les protocoles des réseaux internes des machines parallèles supportés sont essentiellement basés sur le passage de messages, avec des librairies telles que MPL, INX et quelques implémentations spécifiques de MPI.

Les réseaux locaux sont négligés dans l'implémentation actuelle de NEXUS. Il n'est pas possible d'utiliser des protocoles haute performance comme BIP pour MYRINET par exemple. TCP est le seul protocole exploitable avec GLOBUS sur des grappes, et les performances obtenues sont bien en deçà de ce que l'on pourrait espérer si les réseaux rapides étaient directement supportés.

*IRISA, Campus de Beaulieu, 35042 Rennes Cedex, <Alexandre.Denis@irisa.fr>



(a) MADELEINE remplace NEXUS



(b) MADELEINE comme protocole NEXUS

FIG. 1 – *Positions possibles de* MADELEINE *dans* GLOBUS

Objectif Notre objectif est d'étendre GLOBUS pour inclure une prise en charge des réseaux rapides. En effet, les grappes remplacent progressivement les supercalculateurs sur le marché du calcul haute performance grâce à leur meilleur rapport performance/prix. Il n'est pas possible d'utiliser les grappes de manière efficace avec les logiciels de metacomputing actuels car aucun support pour les réseaux haute performance qui équipent les grappes n'y est prévu. Grâce à notre contribution, il est possible d'utiliser efficacement une ou plusieurs grappes dans le cadre du metacomputing, d'interconnecter des grappes, d'utiliser conjointement des grappes et des supercalculateurs, d'exploiter des grappes de manière transparente, de la même façon que l'on utiliserait une grille de supercalculateurs.

Nous étudierons dans une première partie les différentes voix qui s'offrent à nous. Nous nous intéresserons ensuite à la méthode d'intégration. Nous présenterons enfin les performances obtenues, avant de conclure.

2 Intégration de GLOBUS et MADELEINE

MADELEINE MADELEINE [3] est une bibliothèque de communications dédiée spécifiquement aux réseaux haut débit présents sur les grappes. Elle fait partie de l'environnement de programmation parallèle PM^2 [1]. Dans sa deuxième version, elle est capable d'exploiter simultanément plusieurs protocoles parmi SCI, BIP, TCP, VIA et MPI. L'accent est mis sur la performance ; les copies de données sont évitées autant que possible pour atteindre un débit utilisable très proche du maximum autorisé par le matériel.

Pourquoi MADELEINE ? Nous utilisons MADELEINE comme une couche de portabilité. Si nous avions choisi d'inclure de nouveaux protocoles dans NEXUS, il aurait fallu recommencer le travail pour chaque protocole. Grâce à MADELEINE, nous bénéficions immédiatement de tout le travail déjà accompli, et lorsque de nouveaux protocoles sont ajoutés à MADELEINE, GLOBUS en bénéficie automatiquement.

NEXUS/MADELEINE ou GLOBUS/MADELEINE ? Il y avait deux choix possibles pour utiliser MADELEINE à l'intérieur de GLOBUS :

Remplacer NEXUS par MADELEINE. C'est la figure 1(a). Pour cela, on donne à MADELEINE une interface de type NEXUS. Les applications GLOBUS prévues pour utiliser NEXUS utilisent alors MADELEINE sans avoir besoin d'être modifiées. Il ne leur est plus possible d'utiliser NEXUS. Elles sont donc limitées aux topologies autorisées par MADELEINE : une seule grappe. De plus, elles ne sont plus compatibles avec les applications NEXUS.

Intégrer MADELEINE à NEXUS sous la forme d'un nouveau protocole. C'est ce qu'illustre la figure 1(b). Dans ce cas, les applications utilisent toujours NEXUS. NEXUS confie les communications de l'intérieur des grappes à MADELEINE, de façon transparente pour l'application.

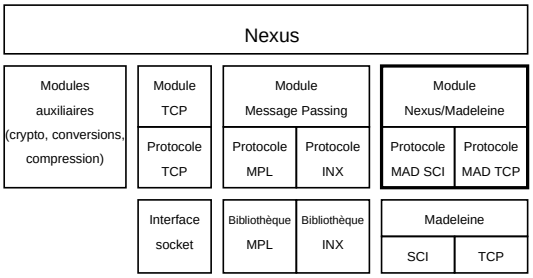


FIG. 2 – *Le module* NEXUS/MADELEINE *au sein de* NEXUS. *La liste des protocoles présentée sur cette figure n'est pas exhaustive.*

NEXUS gère les communications entre les grappes. Cette version autorise des topologies qui intègrent plusieurs grappes. Elle est de plus interopérable avec la version GLOBUS standard présente sur les supercalculateurs.

Notre choix s'est donc porté sur cette dernière possibilité : intégrer MADELEINE comme un protocole NEXUS. Nous appellerons par la suite cette implémentation de NEXUS intégrant MADELEINE "NEXUS/MADELEINE". Ce choix a un inconvénient de taille : on accède à MADELEINE à travers l'interface NEXUS, ce qui introduit inévitablement des copies de données au niveau des couches NEXUS, et donc des performances réduites par rapport à MADELEINE seule.

NEXUS et MADELEINE : deux approches différentes Dans cette section, nous décrivons les approches de NEXUS puis de MADELEINE.

L'approche de NEXUS est totalement dynamique, basée sur la notion de *lien* point-à-point. Les liens sont ouverts et fermés dynamiquement au cours de l'exécution du programme. Le point de départ, appelé *startpoint* est migrable. Il peut être déplacé après sa création. A l'inverse, le *endpoint* est fixe.

L'approche de MADELEINE est quant à elle adaptée à une utilisation sur une grappe. La configuration est statique, régie par un fichier de configuration. Il suffit d'un simple appel à la fonction d'initialisation pour mettre en place toute la topologie. En contrepartie, une fois qu'elle est mise en place, on ne peut pas la modifier dynamiquement. Il est impossible de rajouter ou de retirer un nœud en cours d'exécution. Ce choix est pertinent dans le cas d'une grappe, car la configuration change rarement. En général, on désire utiliser toute la grappe, toujours de la même façon. La simplicité de mise en place est alors un atout.

Les communications sont organisées autour de canaux de communication. Un *canal* au sens MADELEINE est une structure de communication dans laquelle tous les nœuds sont reliés à tous les autres. Un canal est ouvert pour un protocole donné ; il est possible d'ouvrir plusieurs canaux par protocole. L'ouverture est dynamique.

3 Méthode d'intégration : le module NEXUS/MADELEINE

Nous avons fait le choix d'intégrer MADELEINE à NEXUS. Pour parvenir à ce but, il est nécessaire de réaliser un nouveau module NEXUS qui *traduit* les primitives NEXUS en primitives MADELEINE. Il réalise une projection du modèle de communication de MADELEINE (grappe entièrement connectée, topologie statique) vers le modèle de communication de NEXUS (liens point à point, topologie dynamique). La figure 2 montre la place de ce module NEXUS/MADELEINE dans NEXUS.

3.1 Projection des protocoles NEXUS en protocoles MADELEINE

NEXUS sélectionne automatiquement le protocole à utiliser. De plus, il est souhaitable que l'application puisse choisir un protocole manuellement, et donc qu'elle voie les différents protocole MADELEINE à travers NEXUS. Nous choisissons donc de projeter les différents protocoles MADELEINE en différents protocoles NEXUS. Nous ajoutons à la liste des protocoles de NEXUS les protocoles `NEXUS_MAD_TCP`, `NEXUS_MAD_SISCI`, `NEXUS_MAD_BIP` et `NEXUS_MAD_DEFAULT`. Ce dernier protocole NEXUS permet en outre d'accéder à n'importe quel protocole MADELEINE simplement en ajoutant un paramètre optionnel. En l'absence de ce paramètre, le premier protocole MADELEINE est sélectionné. Ces protocoles sont insérés dans la liste de sélection automatique de NEXUS et sont sélectionnés automatiquement par NEXUS quand ils sont disponibles. Même s'ils sont implémentés dans le même module et partagent la plus grande partie de leur code, ces protocoles sont différents et indépendants du point de vue de NEXUS.

Disponibilité Pour mettre en œuvre la sélection automatique, NEXUS a besoin de déterminer pour chaque lien quel protocole est disponible. Pour savoir si un protocole est utilisable sur un lien entre deux machines, NEXUS tente de le créer avec ce protocole. Pour que NEXUS/MADELEINE puisse s'insérer dans les mécanismes de sélection automatique de NEXUS, il est donc nécessaire qu'il soit capable de déterminer si, pour deux nœuds donnés, une connexion MADELEINE est possible. Elle est possible uniquement si les deux nœuds font partie de la même grappe MADELEINE. On fait le choix d'identifier une grappe MADELEINE par l'URL (fournie par MADELEINE) qui décrit le nœud maître. Pour déterminer si une connexion est possible, il suffit donc de comparer les URL des maîtres de deux nœuds considérés.

3.2 Projection des messages

Pour envoyer une requête NEXUS, un module de gestion de protocole reçoit un bloc de données agrégées, prêt à être envoyé. La méthode pour envoyer ce bloc en utilisant MADELEINE est très simple : nous envoyons d'abord la taille des données puis les données elles-mêmes. C'est la méthode canonique pour envoyer rapidement un bloc de données dont la taille n'est pas connue à l'avance.

Optimisation des petits messages Cette méthode d'envoi d'un bloc a un gros défaut : elle utilise systématiquement deux messages MADELEINE. Cela a une influence négligeable sur le débit mais pénalise lourdement la latence. Nous proposons une optimisation qui n'utilise qu'un seul message dans le cas de petits blocs NEXUS, divisant par là leur temps de transfert par deux. La méthode consiste à envoyer un en-tête de taille fixe connue à l'avance qui contient la taille du bloc et une partie des données. Si la totalité des données d'une requête NEXUS tient dans cet en-tête, NEXUS/MADELEINE n'envoie pas de deuxième message MADELEINE. Dans le cas contraire, on envoie un deuxième message MADELEINE ; pour les gros blocs, on retrouve la méthode classique d'envoi en deux messages MADELEINE.

Cette optimisation permet de diviser par deux la latence des petits blocs NEXUS. Pour des blocs de taille inférieure à la centaine d'octets, la latence de NEXUS/MADELEINE sur SCI par exemple reste aux environs de $40\ \mu s$. Avec un seul envoi, la latence arrive à environ $20\ \mu s$. Nous choisissons de dimensionner l'en-tête à 32 octets, taille particulièrement bien adaptée à SCI. 4 octets sont réservés par NEXUS/MADELEINE pour donner la taille du message, NEXUS ajoute un en-tête de 16 octets. Cette méthode est donc capable de transmettre jusqu'à 12 octets de données utiles en un seul envoi. NEXUS/MADELEINE n'utilise alors la méthode à deux envois que pour des blocs de plus de 12 octets de données.

3.3 Projection de la topologie

La figure 3 résume la différence fondamentale qui existe dans la gestion des connexions entre NEXUS et MADELEINE. MADELEINE ouvre systématiquement un canal complet, alors que NEXUS

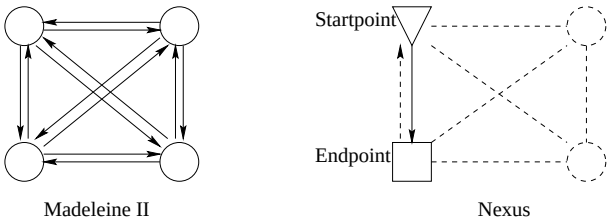


FIG. 3 – *Modèles de communication de* MADELEINE *et* NEXUS

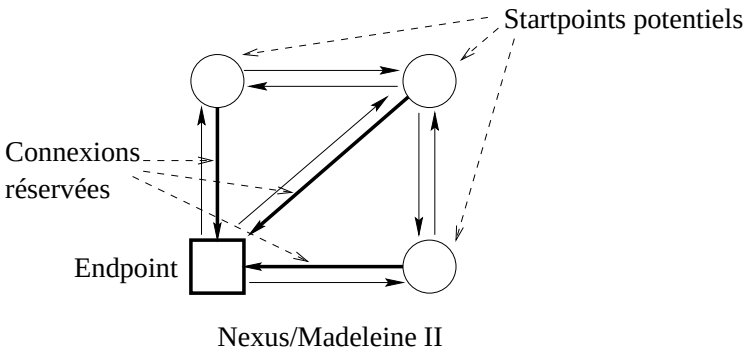


FIG. 4 – *Projection de la topologie* NEXUS *sur la topologie* MADELEINE.

peut n'utiliser qu'un lien à l'intérieur d'une grappe.

Il est donc nécessaire d'établir une projection entre les deux topologies. NEXUS fait passer ses liens sur des canaux MADELEINE. L'approche brutale consiste à ouvrir un canal MADELEINE pour chaque lien NEXUS. Nous sommes ainsi assurés qu'il ne se posera pas de problème lors d'éventuelles migrations du startpoint. Cependant, ouvrir n^2 connexions réseau par lien utilisé sur une grappe de n nœuds conduit à un énorme gaspillage de ressources. Une meilleure approche consiste à tirer profit de l'immobilité des endpoints : puisque seul les startpoints se déplacent, NEXUS peut placer plusieurs de ses liens sur un même canal MADELEINE. Il suffit pour cela de remarquer que seules les liaisons entrantes vers le nœud du endpoint sont potentiellement utilisées. Quand on ouvre un canal, il est possible d'utiliser chaque nœud du canal comme un endpoint NEXUS sans qu'il y ait collision entre les différents liens placés sur ce canal. Ce mécanisme est illustré par la figure 4 : chaque endpoint réserve une partie du canal où il se trouve. Plusieurs startpoints peuvent ainsi se connecter à lui, depuis tous les emplacements possibles.

Avec ce mécanisme, nous gaspillons encore des ressources puisque nous ouvrons un canal (n^2 liaisons) pour n liens. Peut-on aller plus loin ? Il faudrait pour cela autoriser deux endpoints sur le même nœud à partager le même canal, à condition que les startpoints soient sur deux nœuds différents. Cela suppose donc de faire changer le endpoint de canal au cas où la création d'un nouveau startpoint ou une migration créerait une collision. Changer un endpoint de canal implique également un changement de canal pour tous les startpoints qui y sont connectés. C'est techniquement possible, mais contraire à la philosophie NEXUS : un endpoint a peu d'information sur les startpoints qui pointent vers lui, les liens sont à sens unique, justement dans le but d'autoriser des connexions et des migrations de manière très souple.

Cependant, on peut utiliser l'ensemble de façon efficace. Si une application ouvre un lien vers chaque autre nœud de la grappe, à la façon des canaux MADELEINE et si tous les liens entrant en un nœud partagent le même endpoint, alors il n'y a pas de gaspillage de ressources. De cette façon, le récepteur ne connaît pas la provenance des messages — comme dans MADELEINE— et un seul canal MADELEINE est ouvert par NEXUS/MADELEINE.

3.4 Le contrôle

Le canal de contrôle. La gestion des canaux — l'ouverture dynamique en particulier — et la gestion de l'arrêt nécessitent des communications entre les différents nœuds de la grappe MADELEINE. Il nous a semblé tout à fait naturel d'utiliser MADELEINE pour cela. À son initialisation, le module de gestion des protocoles MADELEINE dans NEXUS ouvre un canal dédié au contrôle. Pour ce canal, il choisit le premier protocole disponible. L'efficacité n'est pas primordiale pour ce canal, même TCP suffit amplement. La réception est ensuite gérée de la même façon que sur les canaux de données.

Synchronisation. L'ouverture dynamique de canaux MADELEINE demande une synchronisation globale. Pour cerner le problème, étudions un cas pathologique. Imaginons que deux nœuds demandent l'ouverture d'un nouveau canal simultanément. Ce cas n'est pas improbable, il est même constaté très souvent en pratique en l'absence de mécanisme de synchronisation. Chacun des deux nœuds décide qu'il est nécessaire d'ouvrir un nouveau canal et en informe les autres nœuds. Par malchance, les messages se croisent sur le réseau. Chaque nœud ouvre donc deux nouveaux canaux, ce qui gaspille des ressources si un seul canal aurait suffit. Il y a pire : les deux nœuds qui ont pris l'initiative d'ouvrir un canal ont alors ouvert ces canaux dans un ordre différent. Or la sémantique de MADELEINE demande que chaque nœud ouvre les canaux dans le même ordre, car ils sont identifiés par leur numéro d'ouverture. Ce qui se passe est alors imprévisible et n'est pas spécifié. En pratique, on constate que les canaux sont créés mais ne sont pas connectés comme on attendait.

```
 1: if token owner ≠ me then
 2:    send(msg = REQ TOKEN, dest = token owner)
 3:    while token owner ≠ me do
 4:       wait
 5:    end while
 6:    {Partie en section critique}
 7:    (...)
 8:    {Fin de la section critique}
 9:    token locked ⇐ false
10: end if
```

ALGO. 1: Section critique distribuée.

La solution consiste à vérifier l'ordre de création des canaux. Pour cela, nous choisissons de placer l'ouverture d'un canal dans une section critique. Puisque cette section critique concerne des flots d'exécution répartis sur plusieurs machines, il est nécessaire d'employer un mécanisme d'exclusion mutuelle distribuée. Nous avons choisi d'implémenter l'algorithme de SUZUKI et KASAMI [6], modifié pour tenir compte du multithreading. Cet algorithme a pour principal avantage d'être entièrement distribué. Aucun nœud ne joue de rôle particulier différent des autres. Le principe est simple : un jeton circule, un nœud ne peut prendre la décision d'ouvrir un canal que s'il est propriétaire du jeton. Le jeton n'est libéré que lorsque l'opération d'ouverture est terminée. Le mécanisme exact est décrit par les algorithmes 1 et 2 : le premier est le mécanisme d'entrée et de sortie de section critique ; le second est exécuté en parallèle par NEXUS/MADELEINE sur chaque nœud pour traiter les demandes reçues sur le canal de contrôle. Pour clarifier la présentation, nous n'avons pas indiqué les section critiques locales qui protègent l'accès aux variables.

3.5 Méthodes de réception

Nous avons implémenté trois méthodes de réception. L'une n'utilise pas de threads, les deux autres sont adaptées chacunes à un type de threads.

```
 1: loop
 2:    receive(message)
 3:    if message = REQ TOKEN then
 4:       if token owner = me then
 5:          enqueue(fifo, value = sender rank)
 6:       else
 7:          send(msg = REQ TOKEN, dest = token owner)
 8:       end if
 9:    else if message = GIVE TOKEN then
10:       fifo ⇐ received fifo
11:       broadcast(msg = TOKEN OWNER, value = me)
12:       token locked ⇐ true
13:       token owner ⇐ me
14:    else if message = TOKEN OWNER then
15:       token owner ⇐ received value
16:    end if
17:    if token owner = me and fifo ≠ ∅ and token locked = false then
18:       token owner ⇐ dequeue(fifo)
19:       send(msg = GIVE TOKEN, dest = token owner)
20:       send(msg = fifo, dest = token owner)
21:    end if
22: end loop
```

ALGO. 2: Écoute sur le canal de contrôle.

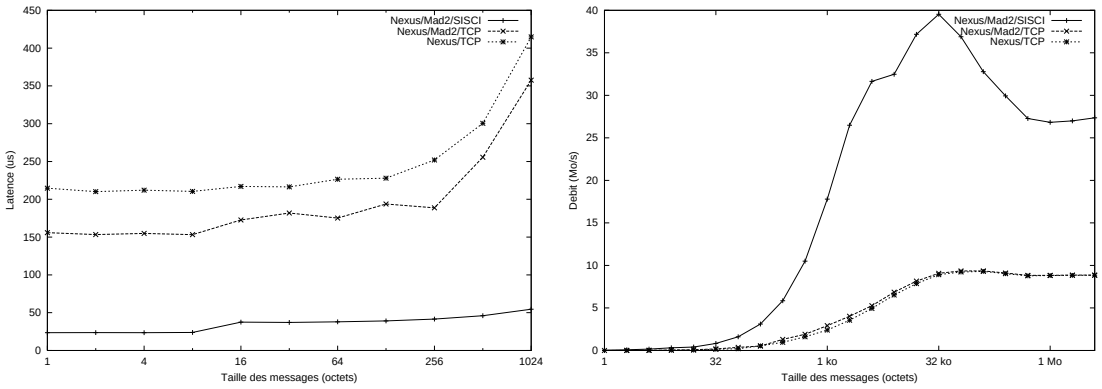
Threads dédiés. Pour obtenir une bonne réactivité, la première version utilise un thread dédié à la réception. Un thread est lancé pour chaque canal sur chaque nœud. Il tourne dans une boucle sans fin qui contient une lecture bloquante sur le canal MADELEINE, puis le traitement du message. Puisque l'appel est bloquant, il ne consomme pas de ressource quand il n'y a pas de message, et le thread est réveillé dès qu'un message arrive (sous réserve que la bibliothèque de threads ait cette possibilité).

Polling sans thread. La première méthode est efficace la plupart du temps, mais ne permet pas l'exploitation de NEXUS/MADELEINE en l'absence de threads. En version mono-threadée, dite également `NEXUS_LITE`, NEXUS gère la réception avec un mécanisme de *callback*. Périodiquement, et à chaque appel explicite de la part de l'utilisateur à la fonction `globus_poll()`, les fonctions enregistrées pour le *callback* sont appelées. Chaque protocole enregistre une telle fonction pour réaliser une scrutation (*polling*) de ses endpoints.

NEXUS/MADELEINE enregistre une fonction de *callback* à son démarrage, cette fonction est appelée périodiquement par NEXUS. À chaque fois qu'elle est exécutée, elle teste l'arrivée de messages sur chaque canal, puis le traite de la même manière que dans le cas d'une version avec threads.

Thread de polling. Cette dernière version est une combinaison des deux précédentes : un seul thread est lancé, il réalise une scrutation en boucle. Il ne requiert pas le lancement de nombreux threads comme la première version ; c'est un avantage dans le cas où l'on ne dispose que d'un nombre limité de threads (cas des threads noyaux).

Discussion. Chaque version a ses avantages et ses inconvénients. La première a pour avantage sa bonne réactivité et pour inconvénient sa grande consommation de ressources systèmes, surtout



(a) Latence (b) Débit

FIG. 5 – *Performances de* NEXUS/MADELEINE

dans le cas de nombreux canaux ouverts. La deuxième est la seule utilisable sans threads, mais a une latence plus élevée (environ $+20\ \mu s$ sur SCI). La troisième a sensiblement les mêmes performances que la version sans thread, mais n'est pas aussi contraignante. Il n'y a pas besoin pour l'application de réaliser une scrutation explicite quand elle attend un message.

On choisira la première version en présence de threads, la deuxième sinon. Cette sélection est réalisée automatiquement par les procédures de configuration et de compilation. L'utilisateur peut forcer l'utilisation, s'il le désire, de l'une ou l'autre des versions à la compilation.

4 Performances de NEXUS/MADELEINE

4.1 Expérimentation

Conditions de test Pour mesurer les performances de notre implémentation NEXUS/MADELEINE, nous avons réalisé un programme qui fait un ping-pong entre deux machines. Il effectue une moyenne sur 10000 allers-retours pour lisser les aléas du réseau. Les machines utilisées sont des bi-processeurs Pentium II 450 MHz, équipés à la fois des interfaces réseaux FASTETHERNET et SCI. Le système est LINUX, avec un noyau SMP. NEXUS seul exploite uniquement TCP sur FASTETHERNET, NEXUS/MADELEINE est testé sur TCP et SCI, avec une réception par threads dédiés (`pthread` linux).

Comparaison sur TCP/IP. Nous comparons dans un premier temps NEXUS/MADELEINE et NEXUS sur TCP/IP au-dessus de FASTETHERNET. Les deux bibliothèques utilisent le même matériel avec le même protocole.

Latence. Comme on le voit sur la figure 5(a), avec $150\ \mu s$ contre $210\ \mu s$, NEXUS/MADELEINE est plus efficace que NEXUS.

Débit. NEXUS/MADELEINE et NEXUS font presque jeu égal avec un maximum à environ 9 Mo/s. C'est pratiquement le maximum autorisé par le matériel FASTETHERNET.

Comparaison des meilleures performances. Nous comparons maintenant les meilleures performances de NEXUS/MADELEINE avec les meilleures performances de NEXUS. On pourrait nous reprocher de comparer des choses qui ne sont pas comparables, mais c'est en fait le type de comparaison que peut faire l'utilisateur final : la vitesse de NEXUS/MADELEINE par rapport à NEXUS

sur le même matériel, peu importe si les méthodes employées sont différentes.

Latence. NEXUS a une latence minimale de 210 μs (en utilisant TCP), NEXUS/MADELEINE a une latence minimale de 23 μs. La possibilité d'exploiter le réseau SCI a divisé la latence minimale par 10. Le gain est très appréciable pour les applications qui échangent de nombreux petits messages.

Débit. NEXUS/MADELEINE monte jusque 40 Mo/s grâce à l'utilisation de SCI, à comparer aux 9 Mo/s de NEXUS limité à TCP (figure 5(b)). Le débit de NEXUS/MADELEINE redescend à 30 Mo/s quand l'effet de la mémoire cache s'estompe. S'il n'y avait pas de recopie systématique, le débit continuerait à augmenter.

4.2 Analyse des performances

Le poids des copies. À cause des copies introduites par NEXUS, le débit est fortement pénalisé. Les performances en crête de NEXUS/MADELEINE sur SCI n'arrivent qu'à 50 % du débit maximal dans MADELEINE sans GLOBUS. Quoique décevantes à première vue, ces faibles performances sont explicables. NEXUS effectue une copie lors de l'émission pour agréger les différentes parties d'un message. Cette copie est inévitable car la sémantique de NEXUS autorise l'application à modifier les données après les avoir empaquetées sans que cela ait de conséquence sur les données envoyées. Il effectue une copie en réception car les données étant envoyées de manière agrégées, même si aucune conversion de type n'est nécessaire, elles ne sont jamais reçues directement dans l'espace de stockage de l'application.

Posons D_{mem} le débit de copie en mémoire, et D_{net} le débit du réseau. Puisque NEXUS envoie les données en un seul bloc, il n'y a aucun recouvrement entre les recopies en mémoire et les transmissions sur le réseau. Le débit maximal de la séquence copie mémoire, transmission sur le réseau, copie mémoire a donc pour débit D, avec D défini par :

$$D = \frac{1}{\frac{2}{D_{\text{mem}}} + \frac{1}{D_{\text{net}}}}$$

Prenons $D_{\text{net}} = 75$ Mo/s et $D_{\text{mem}} = 160$ Mo/s (valeurs mesurées expérimentalement), nous obtenons $D = 38$ Mo/s. Les performances de NEXUS/MADELEINE sont donc très proches de leur maximum théorique. Le débit maximum mesuré dépasse même cette valeur ; il faut attribuer ceci à la vitesse des copies en mémoire qui est bien plus rapide quand elle a lieu à l'intérieur de la mémoire cache du microprocesseur.

Un gain appréciable. Malgré cette pénalité introduite par les copies, le gain de performance de NEXUS à NEXUS/MADELEINE est tout de même appréciable. Grâce à l'exploitation du réseaux SCI, le débit est multiplié par 3 et la latence divisée par presque 10. Même en se limitant à TCP dans l'hypothèse d'une grappe équipée uniquement du réseau FASTETHERNET, NEXUS/MADELEINE a de meilleures performances avec notamment une meilleure latence. Ceci s'explique aisément par le fait que NEXUS a une implémentation TCP optimisée pour les longues distances où la latence n'a que peu d'importance, alors que MADELEINE cible spécifiquement les réseaux haut débit. Ce gain de performance est obenu d'une manière totalement transparente pour l'application : exactement le même code source est utilisé pour compiler cette application avec NEXUS et NEXUS/MADELEINE.

5 Conclusion

Bilan. L'implémentation NEXUS/MADELEINE permet à GLOBUS d'utiliser efficacement des grappes. Il est maintenant possible de combiner la puissance de la partie d'administration et de gestion de ressources de GLOBUS avec la performance de MADELEINE sur les réseaux haut débit. GLOBUS est maintenant capable d'exploiter des réseaux SCI et TCP/IP de manière optimisée pour les grappes, bientôt MYRINET avec le protocole BIP.

Cette utilisation de nouveaux protocoles est généralement entièrement transparente : la même application est capable d'utiliser GLOBUS avec ou sans MADELEINE, et les deux versions sont interopérables. La seule différence notable est une accélération des communications sur le réseau quand elles restent à l'intérieur de la grappe.

Travaux futurs. Il reste encore de nombreuses améliorations à apporter à NEXUS/MADELEINE en particulier, et à GLOBUS en général, pour que le support des grappes soit réellement efficace. Il faudrait par exemple intégrer au gestionnaire de ressources GRAM les mécanismes de lancement de processus de MADELEINE dans le cas des protocoles qui nécessitent un chargeur d'application externe (BIP, MPI).

Ces travaux ont permis de révéler les faiblesses de l'API de NEXUS. Les communications zéro-copie ne sont pas encore supportées, alors que l'interface a été prévue depuis longtemps. Malheureusement les efforts de développement actuels de l'équipe GLOBUS ne vont pas dans cette direction. Cette interface est pourtant indispensable pour obtenir des performances correctes sur des réseaux haut débit. Il est impossible pour les modules de protocole d'effectuer des transmission pipelinées à cause encore une fois de l'API qui ne l'autorise pas : les données sont systématiquement agrégées avant d'être disponibles pour le protocole.

La plupart de ces imperfections proviennent du fait que la distinction entre les liaisons longue distance où la performance n'est pas essentielle et les liaisons locales haut débit n'a pas été faite. Il manque à NEXUS une connaissance de la hiérarchie du réseau pour qu'il puisse adapter ses méthodes au type de lien : il pourrait ainsi par exemple activer automatiquement les mécanismes zéro-copie sur les liens haut débit et la conversion de données uniquement sur les liens qui sortent de la grappe ou du supercalculateur. Une modification de NEXUS pour prendre en compte la hiérarchie est tout à fait envisageable étant donné le fait que cette hiérarchie est déjà connue par les modules de gestion de ressources. Il serait également souhaitable d'intégrer à NEXUS un mécanisme qui permet un envoi pipeliné sur le réseau. Il suffirait de connaître le destinataire dès l'initialisation du buffer d'envoi.

NEXUS n'a pas beaucoup évolué au cours de ses quatre précédentes versions. Les modifications que nous proposons sont par contre une complète refonte de la conception des communications dans le cadre du metacomputing. Auparavant le *metacomputer* était vu comme un ensemble de nœuds tous égaux. Afin d'améliorer la localité des applications, il est nécessaire de prendre en compte la hiérarchie du réseau. Le modèle "à plat" doit donc évoluer vers une approche "grappe de grappes".

Remerciements

Je tiens à remercier les membres du projet PM^2 (LIP, ENS Lyon) pour leur support technique et leur accueil.

Références

[1] PM2 High Perf. World Wide Web document, `http://www.pm2.org`.

[2] The Globus Project. World Wide Web document, `http://www.globus.org`.

[3] Olivier Aumage, Luc Bougé, and Raymond Namyst. A portable and adaptive multi-protocol communication library for multithreaded runtime systems. Research Report RR2000-17, LIP, ENS Lyon, Lyon, France, April 2000.

[4] Ian Foster, Jonathan Geisler, Carl Kesselman, and Steven Tuecke. Managing multiple communication methods in high-performance networked computing systems. *Journal of Parallel and Distributed Computing*, 40(1):35–48, 1997.

[5] Ian Foster and Karl Kesselman, editors. *The Grid: Blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers, 1999.

[6] M. Raynal. *Algorithmique du parallélisme*, chapter 5 : Le problème de l'exclusion mutuelle en distribué, pages 122–128. Dunod Informatique, 1984.