Mise en oeuvre et évaluation de l'éclatement des connexions TCP pour optimiser l'exécution des applications MPI sur une grille

Stéphane Alter, Olivier Glück

Laboratoire de l'Informatique du Parallélisme

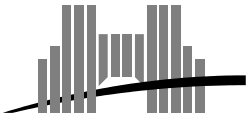
École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

Mise en oeuvre et évaluation de l'éclatement des connexions TCP pour optimiser l'exécution des applications MPI sur une grille

Stéphane Alter, Olivier Glück Juin 2009

Research Report N° RR-6986

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr



Mise en oeuvre et évaluation de l'éclatement des connexions TCP pour optimiser l'exécution des applications MPI sur une grille

Stéphane Alter, Olivier Glück

Juin 2009

Abstract

The MPI standard is often used in parallel applications for communication needs. Most of them are designed for homogeneous clusters but MPI implementations for grids have to take into account heterogeneity and long distance network links in order to maintain a high performance level. These two constraints are not considered together in existing MPI implementations. MPI5000 is a transparent applicative layer between TCP and MPI, using proxies to improve the execution of MPI applications on a grid by spliting connections. We proposed a performance evaluation by reproducing MPI communications to set an execution environment close to reality. We studied the impact of MPI5000 on an MPI_Gather application (all to one) concerning the execution time, the number of retransmissions and timeouts, and the CPU usage. The experiments have been performed on the Grid'5000 French national platform.

Keywords: MPI5000, MPI, grid, WAN, Grid'5000

Résumé

Les applications parallèles utilisent généralement le standard MPI pour réaliser leurs communications et s'exécutent aujourd'hui sur des grilles de calcul. Aucune implantation actuelle de MPI ne prend en compte efficacement les contraintes des connexions longue distance permettant l'interconnexion des sites de la grille. MPI5000 est une proposition d'architecture placée entre TCP et MPI permettant l'éclatement des connexions TCP de manière transparente. Celle-ci permet par l'intermédiaire de proxy à l'interface LAN-WAN de différencier les deux types de trafic. Nous avons notamment étudié l'impact de MPI5000 sur une application de MPI_Gather (tous vers un) en regardant le temps d'exécution, le nombre de retransmissions et de timeouts ainsi que l'utilisation CPU. Les expérimentations ont été réalisées sur la grille française Grid'5000.

Mots-clés: MPI5000, MPI, grille, longue distance, Grid'5000

Sommaire

Table des figures

Introduction

Les besoins en puissance de calcul intensif sont très importants de nos jours, notamment pour la simulation. Une des architecture étant capable de répondre à ces besoins est la grille de calcul, une architecture en plein développement. Elles consistent en un réseau d'ordinateurs faiblement couplés et ont pour but d'offrir une très grande puissance de calcul à leurs utilisateurs de la façon la plus transparente possible. Ces ordinateurs peuvent être des super-calculateurs, des clusters ou des stations de travail ordinaires. Ils sont reliés par un réseau à très grande échelle, le plus souvent l'internet. Cependant, pour qu'une application puisse s'exécuter sur plusieurs machines, il est nécessaire de communiquer entre les différents processus qui exécutent chacun une partie du calcul. Cette communication peut se faire grâce une interface de programmation à passage de messages telle que Message Passing Interface (MPI). Il est nécessaire de prendre en compte les caractéristiques différentes des réseaux locaux et longue distance. Les attentes de messages étant un facteur limitant pour une application parallèle, la prise en compte ces paramètres dans l'élaboration d'une solution d'optimisation est indispensable.

Dans la suite du rapport, nous allons tout d'abord définir le contexte des applications parallèles et des grilles de calcul. Dans un second temps, nous verrons les enjeux concernant l'optimisation des communications afin de réduire le temps d'exécution des applications ainsi que les questions permettant d'avancer dans cette voie. Dans la partie suivante, nous analyserons une implantation existante nommée MPI5000 permettant de répondre partiellement à nos problématiques. Ensuite, nous présenterons nos travaux visant à quantifier les performances de MPI5000 dans le but d'identifier les points faibles et améliorations possibles. Nous terminerons par les conclusions concernant ce travail et les perspectives associées.

Contexte

1 Les applications parallèles

Une application scientifique parallèle, s'exécutant sur une grappe de calcul, effectue des phases de calcul et de communication. Ces phases sont souvent bien distinctes dans le sens ou un processus transite d'une phase de calcul vers une phase de communication et vice versa. Les phases de communications peuvent bloquer le processus dans le cas d'une communication synchrone ou bien être non-bloquante dans le cas d'une communication asynchrone. Ainsi, chaque tâche de l'application oscille entre des phases de calcul et des phases de communications bloquantes ou non bloquantes. Le temps d'exécution du programme dépend donc à la fois de la puissance de calcul et de la vitesse des communications réseaux. Les applications parallèles se servent de MPI pour assurer la communication entre les tâches.

2 Le calcul parallèle avec MPI

MPI (The Message Passing Interface) a été conçue en 1993-94. C'est une norme définissant une bibliothèque de fonctions, utilisable avec les langages C, C++ et Fortran. Elle permet d'exploiter des ordinateurs distants ou multiprocesseur par passage de messages.

Les tâches d'une application parallèle ont besoin de communiquer pour s'échanger les données nécessaires à l'avancement de chacune des tâches. Pour cela, elles vont utiliser des primitives de communication fournies par l'API MPI. Il s'agit typiquement des fonctions d'envoi et de réception. Pour pouvoir se synchroniser, les processus vont utiliser différentes fonctions comme des fonctions de communications point à point (MPI_Send, MPI_Receive) ou des fonctions de communications collectives. Celles ci permettent de communiquer une information de 1 vers n, de n vers 1 ou de n vers n processus.

Il existe plusieurs implantations de MPI, certaines sont propriétaires et d'autres libres. Une étude comparative d'implantations MPI dans grid'5000 a été réalisée [8].

3 Les grilles

Les grilles informatiques sont des plates-formes de calcul à grande échelle, hétérogènes et distribuées. Les grilles que nous considérons sont des infrastructures composées de ressources de calcul, de stockage et de communications dédiées au calcul haute performance.

Grid'5000 [2] est une grille expérimentale de recherche. Ce genre de grille n'offre pas de garantie de service, dans la mesure ou des modifications peuvent être apportées sur l'infrastructure ou le système d'exploitation. Elle est composée de sites géographiques constitués par des grappes de ressources interconnectées par des réseaux rapides. Les connexions entre les sites sont assurées par RENATER[13] grâce à des liens isolés du reste du trafic internet. La spécificité des liens longue distance nous place dans le contexte des *LFN*[1]. Ce sont des liens qui ont un produit Délais de transmission multiplié par le Débit très élevé. La figure 1 représente la topologie des liens longue distance appartenant à la grille Grid'5000.

[1] LFN = long and fat networks

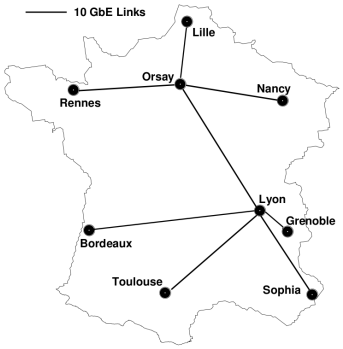


Figure 1: Liens inter-sites de Grid'5000

4 Les caractéristiques des réseaux

Finalement, on peut voir la grille comme une interconnexion de clusters (grappe de PC) par un réseau longue distance (WAN). Les réseaux de la grille sont hétérogènes. D'une part, il existe des réseaux rapides, aussi appelés LAN, permettant l'interconnexion des nœuds dans les clusters qui utilisent des technologies diverses telles que Myrinet, Infiniband ou Gigabit ethernet.

D'autre part, l'interconnexion entre les sites est de type fibre optique, offrant des débits de 10 Gb/s. Le temps minimal de réponse entre les sites varie de 3ms à 20ms en fonction de l'éloignement des sites. Une évaluation des liens de Grid'5000 est proposée dans [6].

Chaque site possède du matériel et une topologie réseau particulière. Toutes les informations sont disponibles sur le wiki de Grid'5000[3]. La figure 2 est tirée du wiki de Grid'5000 et représente la topologie réseau du site de Bordeaux.

Nous pouvons distinguer les différents clusters : *bordemer*, *bordeplage*, *borderline*, *bordereau* sont connectés à Renater par un réseau éthernet. Les nœuds de *bodemer* sont connectés entre eux par un réseau Myrinet 2G, ceux de *bordeplage* par un réseau Infiniband 10G, ceux de *borderline* par un réseau Myrinet 2G ainsi qu'un réseau Infiniband 20G.

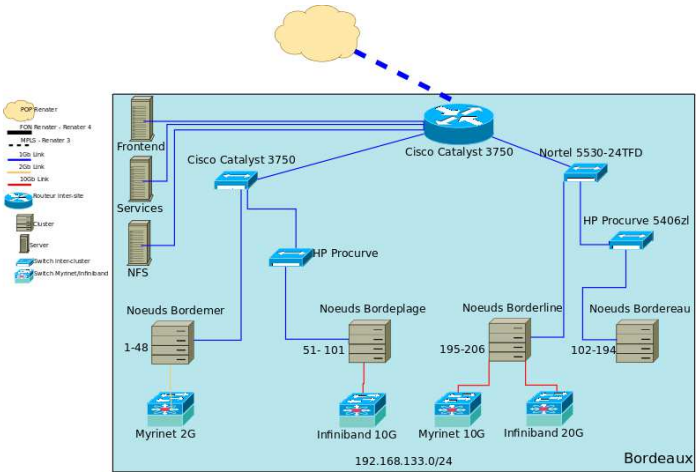


Figure 2: Topologie réseaux du site de Bordeaux

Problématique

Nous nous plaçons dans le contexte d'exécution d'applications parallèles de type MPI sur une grille de calcul. Ces applications utilisent les réseaux LAN (au sein d'un même site) et WAN (entre les sites) de la grille pour communiquer et leur performance est directement liée à la progression des communications. La problématique générale est donc d'optimiser les communications afin de réduire le temps d'exécution de l'application sur ce type de plateforme en optimisant les communications. L'enjeu principal est de proposer des mécanismes de communication assurant un échange de message rapide entre les nœuds de calcul pouvant se situer à la fois sur le même site géographique ou bien distants de plusieurs centaines de kilomètres. En effet, la performance d'une application est directement liée à la progression des communications. Sur une architecture de type grille, l'application est pénalisée par les communications longue distance. Les kilomètres séparant les différents sites de la grille étant incompressibles, la latence des communications inter-sites restera toujours élevée. Néanmoins, avec les implantations actuelles de MPI, les applications ne font pas la différence entre leur exécution sur un cluster ou sur une grille. Face à notre problématique, nous nous demandons si le temps d'exécution de l'application pourrait être amélioré en lui faisant prendre conscience de la topologie de la grille sur laquelle elle s'exécute. Ainsi nous considérons que les connexions entre les nœuds peuvent être éclatées de la manière suivante. En introduisant un nœud possédant une fonction spécifique de regroupement de connexions issues d'un même site, il est possible de transformer une connexion LAN-WAN-LAN en trois connexions. Une connexion du nœud vers la passerelle de son site (LAN-LAN), une entre deux passerelle sur deux sites distants (WAN-WAN) et enfin une de la passerelle vers le nœud destinataire (LAN-LAN). L'éclatement des connections est une solution permettant de proposer des optimisations différentes adaptées aux caractéristiques différentes du WAN et du LAN.

Une application parallèle faisant intervenir un grand nombre de nœuds est susceptible de créer de la congestion. En effet, grâce aux fonctions MPI de communication collective, il est courant de dépasser le débit du lien WAN et saturer les commutateurs.

Est-ce bénéfique de faire prendre conscience à l'application de l'infrastructure de la grille, et donc bénéfique d'éclater les connexions ?

Sans congestion, les passerelles seules, sans optimisation complémentaires du lien WAN n'apportent rien en matière de temps gagné. Qu'apportent les passerelles dans un contexte congestionné ? Est-ce vérifié expérimentalement ?

On suppose qu'une telle architecture permet de limiter les pertes sur le WAN. En effet, le fait d'avoir une seule connexion longue distance permet de ne pas dépasser le débit théorique du lien. Quelles sont les pertes et les retransmissions au niveau des passerelles, au niveau des nœuds ? Est-ce que les passerelles permettent effectivement de réduire le nombre de paquets retransmis sur le WAN ? La stratégie de limiter les retransmissions longue distance permet-elle de minimiser le temps global d'exécution ?

La taille des messages MPI a-t-elle une influence sur les performances de MPI5000 ? Si oui, pour quelles tailles de messages le rendement des passerelles est-il optimal ?

Enfin, les passerelles représentent un goulot d'étranglement dans la mesure où tous les messages issus d'un site passent par elle. La vitesse de traitement des messages de ces passerelles est-elle une limite au bon fonctionnement de MPI5000 ? Si oui, à partir de combien de nœuds et pour quelle taille de message ?

Etat de l'art

Dans cette partie, nous allons présenter MPI5000, la couche applicative permettant de mettre en oeuvre l'éclatement des connexions. Ensuite, nous verrons les différents type de communications et méthodes d'envoi de messages dans MPI. Par ailleurs, nous présenterons le protocole de transport TCP utilisé notamment sur le WAN. C'est à ce niveau là que des améliorations sont possibles pour gagner en vitesse de communication et donc en temps d'exécution. Nous verrons ensuite une application nommée Web100 permettant d'accéder aux informations propres à TCP. Elle nous sera utile dans les expérimentations pour relever le nombre de paquets perdus par les nœuds et les passerelles.

5 Communication dans MPI

Comme vu précédemment, MPI est une norme qui propose un standard d'envoi de messages pour les applications parallèles. Il existe différentes primitives de communication prédéfinies pour permettre d'effectuer tout type de communication entre les nœuds effectuant les calculs.

5.1 Les types de communication

Les trois principaux types de communication proposés par MPI sont les suivants:

- Point à Point
- Un vers Tous
- Tous vers Un

Nous allons nous intéresser plus particulièrement dans la suite du rapport à des communications collectives de type : Tous vers Un. En effet, en plaçant stratégiquement le nœud destinataire sur un site géographique, et tous les autres voulant communiquer vers lui à partir un autre site, nous voulons provoquer de la congestion interne à l'application de manière maîtrisée à la fois sur le LAN ainsi que sur le WAN. Ainsi, il sera possible d'avoir une analyse précise des comportements de MPI avec passerelle. Nous verrons plus loin dans la section contribution les choix que nous avons fait pour réaliser nos expérimentations.

5.2 Les messages

5.2.1 Les modes d'envoi

MPI prévoit deux types de mode d'envoi de messages selon leur taille. Le mode eager pour les petits messages et le mode rendez-vous pour les gros messages. La taille des données à envoyer détermine l'utilisation du protocole de rendez- Vous. Néanmoins, la valeur de la taille, provoquant le changement de protocole n'est pas définie par le standard MPI, elle dépend des choix de l'implantation MPI selon les particularités propres des différents réseaux.

mode eager : Les données sont envoyées en même temps que la requête initiant la demande de communication.

mode rendez-vous : Il consiste à envoyer un premier message de petite taille pour préparer la tâche réceptrice et lui permettre d'allouer la mémoire nécessaire.

Dans nos expérimentations nous allons faire varier la taille des messages pour évaluer les performances de MPI5000 dans des contextes les plus diversifiés possibles. La connaissance de la valeur du seuil nous permettra de comprendre les résultats obtenus.

5.2.2 Les méthodes d'envoi

MPI propose différentes méthodes d'envoi des données :

- bloquant / non bloquant
- synchrone / asynchrone

En effet, un envoi non-bloquant (MPI_Isend) ne transfère pas immédiatement les données sur le support physique. Il redonne la main à la tâche MPI avant même la terminaison de la communication. Ces méthodes asynchrones permettent un recouvrement des communications par le calcul, en particulier, si l'implantation MPI contient un ou plusieurs threads dédiés aux communications.

Toutefois, la tâche exécutant une méthode non-bloquante devra faire appel à une méthode d'attente (MPI_Wait ou MPI_Test) pour s'assurer de la terminaison de la communication.

A l'inverse, un envoi bloquant (MPI_Send) initie la communication au moment de l'appel de la méthode d'envoi. Il rend la main à la tâche uniquement après que la communication soit terminée, c'est à dire à partir du moment où le tampon contenant les données d'envoi peut-être réutilisé. En outre, certaines implantations (comme MPICH) utilisent plusieurs méthodes d'envoi suivant la taille des messages à envoyer.

Par exemple, pour des messages de petites tailles, un MPI_Send rend la main après une copie du tampon mémoire et non à la fin de la communication. Pour de grands messages, la même opération MPI_Send utilise un mode synchrone qui attend un message confirmant que les données peuvent être réceptionnées. Elle se termine lorsque toutes les données ont été transférées vers la librairie bas-niveau.

6 TCP

TCP (Transmission Control Protocol) est un protocole de transport fiable, en mode connecté. Les deux hôtes d'extrémité sont responsables du débit de transfert des données. Ils décident à quel vitesse transmettre et quand accélérer et décélérer le débit ainsi que quand transmettre. Le réseau ne leur fournit pas d' informations explicites.

C'est le protocole de transport utilisé pour les connexions inter-sites, sur le WAN. C'est à ce niveau que des optimisations sont possibles, notamment en jouant sur l'"agressivité" du protocole.

Il existe deux mécanismes de TCP qui impactent sur les applications MPI : le contrôle d'erreur ainsi que le contrôle de congestion.

6.1 Contrôle d'erreur dans TCP

TCP sauvegarde chaque paquet envoyé dans un tampon en attendant de recevoir un acquittement (ACK) du destinataire. Si le paquet est perdu, TCP le retransmet. Il existe deux cas de détection de perte. Si il y a assez de segments à transmettre après la perte, le premier cas consiste à attendre 3 duplicate Acks (dupAcks) d'un segment pour retransmettre le segment suivant en activant l'algorithme de Retransmission Rapide (Fast Retransmit). L'attente avant retransmission est de l'ordre d'un RTT[2]. Dans le second cas, si il n'y a plus rien a transmettre après la détection de la perte, il faut alors attendre l'arrivée d'un timeout (RTO) avant de pouvoir retransmettre le paquet perdu. Ce RTO dépend du RTT et est beaucoup plus grande qu'un RTT (généralement 200ms + RTT sous linux). Ce cas est donc très pénalisant pour les applications MPI.

6.2 Contrôle de congestion dans TCP

La congestion d'un réseau informatique est la condition dans laquelle une augmentation du trafic[3] provoque un ralentissement global de celui-ci. Les paquets entrant dans les buffers des commutateurs sont rejetés dans ce cas. Une analyse du comportement de certains commutateurs dans le contexte de réseaux rapides avec congestion est présente dans [14]. En général, la perte d'un paquet est la seule information sur l'état du réseau. Le contrôle de congestion est géré exclusivement par l'émetteur. Il consiste à limiter la vitesse d'émission dans le cas de perte de paquets dues à la congestion. Le récepteur ne fait que renvoyer des accusés de réception.

TCP utilise une fenêtre de congestion, qui détermine la quantité de données qui peut être envoyé en même temps sur un lien. La Figure 3 montre l'évolution de la fenêtre de congestion. Afin de déterminer la bande passante d'un lien, TCP utilise un premier un mécanisme nommé slowstart dans lequel la première fenêtre de congestion est fixée d'abord à deux paquets et augmente de manière exponentielle à chaque réception d'un ACK jusqu'à ce qu'il y ait une perte. Ensuite, en mode stabilisé, cette fenêtre augmente de façon linéaire. Quand une perte se produit (soit détecté par trois dupAcks ou un timeout), la fenêtre de congestion diminue. Enfin, après un temps d'inactivité (rien n'est envoyé sur une connexion pour une longue période), TCP entre de nouveau dans une phase de slowstart.

Dans les réseaux filaires, la congestion prend racine dans la destruction ou la rétention des paquets au niveau des buffers (seulement 1% des pertes de paquets dans l'Internet est dû à l'altération du contenu). La fenêtre de congestion doit donc être en mesure de s'adapter à l'état courant des buffers, et ce sans surestimer la congestion au risque d'influer négativement sur le débit de la connexion.

[2] RTT : Round-trip delay time. C'est le temps nécessaire à un paquet pour faire un aller-retour entre la source à la destination

[3] Flux

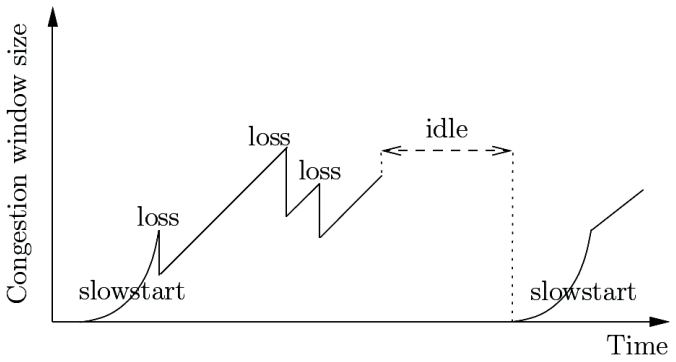


Figure 3: Evolution de la fenêtre de congestion TCP

6.3 Paramétrage de TCP

Un des objectifs de MPI5000 étant de rendre possible la différentiation des protocoles entre le LAN et le WAN, il est intéressant d'étudier les différents protocoles de transport pouvant satisfaire les contraintes de notre architecture imposée par la grille. Dans cette optique d'optimisation des communications, nous allons tout d'abord nous focaliser sur les communication du WAN. Ce type de lien est qualifié de *high BDP*[4], signifiant haute valeur du produit $bandwidth * RTT$. Une étude des adaptations de TCP pour ce type de réseau à réalisée [11].

La taille par défaut des tampons TCP dans les systèmes d'exploitation est prévue pour des réseaux lents à faible latence. Il est maintenant connu que la taille des tampons d'émission/réception d'une connexion TCP doit être supérieure au produit du temps aller- retour (RTT) séparant la source de la destination par le débit théorique du lien, et ce afin de pouvoir profiter pleinement des performances du réseau. Ceci permet de ne pas être bloqué par la dimension du tampon et de s'assurer qu'il aura potentiellement toujours plus de données à envoyer que le lien ne peut en écouler. Dans notre cas, il faut prévoir une taille de tampon suffisamment grande pour le WAN (4Mo).

Par ailleurs, on peut envisager d'utiliser sur les passerelles de MPI5000 une variante de TCP mieux adaptée aux LFN.

6.4 Variantes du protocole TCP

Une analyse comparative des différentes variantes de TCP a été réalisée par des membres de l'INRIA dans [5]. Cet article présente une étude comparative des variantes de TCP suivantes soumises à différentes latences :

- TCP Reno - évolution lente de la fenêtre de congestion (non adapté aux haut BDP)
- BIC
- CUBIC
- HS-TCP (HighSpeed TCP)

[4]BDP = Bandwidth*Delay Products

- H-TCP (Hamilton TCP)
- Scalable TCP

Les simulations montrent que Reno n'est pas du tout adapté aux réseaux à forte BDP. D'après les résultats, si le seul objectif est de maximiser le débit, il faut adapter la variante de TCP en fonction du RTT de la manière suivante :

BIC si $RTT < 20ms$

HighSpeed TCP si $20ms < RTT < 150ms$

H-TCP si $RTT > 150ms$

Ces résultats nous informent que la meilleure solution dans notre cas est d'utiliser BIC pour les passerelles de MPI5000. En effet, dans le cas de Grid'5000, les RTT les plus élevés entre deux sites géographiques ne dépassent pas 20ms.

Par ailleurs, il existe une technique nommée *Tcp pacing* [1] [15], qui consiste en un espacement des paquets dans le but de prévenir une congestion au niveaux des commutateurs. Cette technique est utilisée dans le but d'éviter l'envoi de paquets en rafale, chose qui peut provoquer la saturation des commutateurs et se traduire par des rejets de paquets. Il peut être intéressant de l'intégrer dans MPI5000 sur les nœuds pour limiter les pertes dues aux rafales.

Enfin, d'autre protocoles ont été étudiés pour répondre aux problèmes rencontrés avec l'utilisation de TCP sur les liens de type *High BDP*. Par exemple SCTP[9], UDT[4] ou XCP[12]. L'implantation de tels protocoles est envisageable entre les passerelles de MPI5000 pour permettre de gagner en efficacité sur le lien WAN.

7 MPI5000

MPI5000 est une proposition d'architecture permettant l'éclatement des connexions de manière transparente pour une application parallèle s'exécutant sur la grille. MPI5000 a été développé par Ludovic Hablot au LIP.

7.1 Présentation

MPI5000 [7] a pour but de contrôler et d'améliorer les communications MPI sur les grilles. En effet, c'est une couche transparente permettant d'exécuter une application MPI en utilisant des passerelles. Ces relais ont une vision globale des communications sur les liens longue distance. Nous allons évaluer cet outil dans la suite du rapport. Sur la figure 4, on voit en pointillés les connexions sans MPI5000 alors que les connexions avec MPI5000 sont représentées par des lignes pleines. $N_{x.y}$ représente le nœud numéro y sur le site numéro x. $P_{x.y.z}$ représente le processus numéro z sur le nœud numéro y sur le site numéro x. G_x représente la passerelle du site numéro x. On remarque que sans MPI5000, chaque processus établit une connexion TCP avec tous les autres processus, quelque soit leur localisation. Tandis qu'avec MPI5000, tous les processus créent une connexion avec la passerelle de leur site, puis les passerelles créent une seule connexion entre elles.

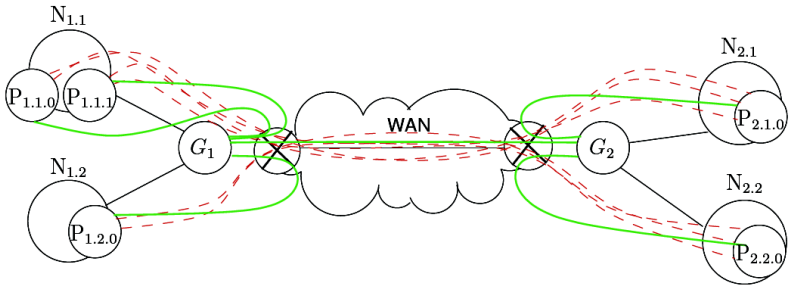


Figure 4: Éclatement des connexions

De nombreux avantages sont apportés comme restreindre l'éventuelle congestion au LAN, une meilleure signalisation de la congestion externe car on obtient moins de timeouts et plus de DupAcks. MPI5000 permet de ne subir le slowstart que sur une seule connexion longue distance et de réduire la taille des buffers TCP sur les nœuds. MPI5000 ne crée qu'un seul lien entre les sites. Cela permet aussi de ne pas dépasser le débit du lien. Enfin, elle permet d'envisager une différentiation des protocoles entre le LAN et le WAN.

D'autre part, MPI5000 de par sa structure introduit un surcoût égal à 2 RTT locaux supplémentaires pour chaque communication ainsi que deux copies supplémentaires. En effet, une recopie Carte vers Tampon TCP et Tampon TCP vers Tampon niveau utilisateur lors de la réception, et inversement lors de l'envoi. Sur la figure 5 les traits pleins représentent le cheminement des messages sans MPI5000, tandis que les traits en pointillés sont avec MPI5000. On constate le surcoût de 2 RTT.

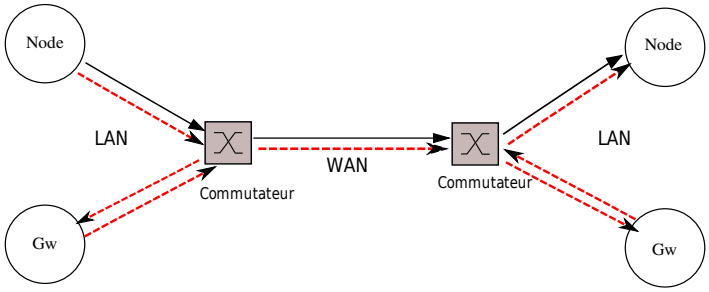


Figure 5: Surcoût dû aux passerelles

Nous allons dans la suite de ce rapport mettre en avant les scénarios qui apportent un gain, ainsi que quantifier et localiser les retransmission dues à la congestion.

7.2 Architecture et fonctionnement

Sur la figure 6, la ligne en pointillés représente le chemin habituel de communication. Cette architecture sépare cette connexion en deux connections *locales* et une connexion *longue distance*. Sur les nœuds, une librairie est chargée de manière dynamique et transparente (LD_PRELOAD) lors de l'exécution d'une application parallèle. Sur les passerelles est exé-

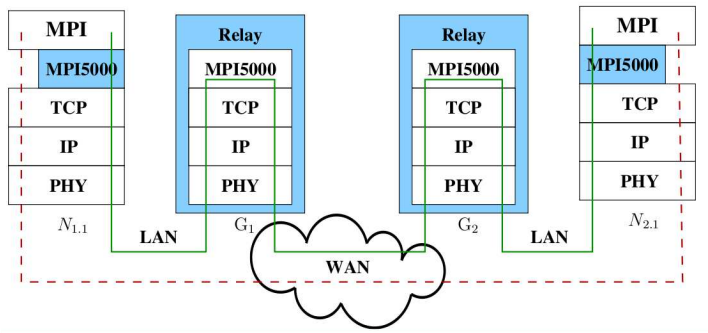


Figure 6: Architecture de MPI5000

cuté un démon MPI000d qui joue le rôle de proxy et se charge de retransmettre les messages du LAN vers le WAN et du WAN vers le LAN.

8 Web100

Web100 [10] est une suite de programmes destinée aux machines linux qui permettent l'analyse profonde d'une connexion TCP à la volée. Grâce à l'application d"un patch au noyau linux, Web100 instrumente la pile TCP en rendant visible un certain nombre de variables TCP, comme la fenêtre de congestion, le seuil "ssthresh", le RTT, le nombre d'appel à la fonction FastRetransmit, le nombre de timeout, la fréquence ou la quantité de données envoyées/reçues.

Nous allons utiliser Web100 pour récolter des informations sur les Timeouts et les DupAcks reçus à la fois au niveau des nœuds, mais aussi au niveaux des passerelles de MPI5000. Ainsi, nous pourrons quantifier séparément les retransmissions sur le LAN et sur le WAN lors de l'utilisation de MPI5000.

Contributions

9 Rappel du problème et définition du scénario d'étude

Nous considérons le cas de l'utilisation de MPI5000 dans le but de réduire le temps d'exécution d'une application parallèle échangeant des messages MPI sur une grille. L'éclatement des connexions entre les nœuds est réalisé par MPI5000. Cet éclatement provoque un surcoût en terme de temps de transfert des messages d'un nœud à un nœud d'un autre site. Ce surcoût est expliqué et quantifié dans [7]. Mais un des atouts des passerelles est de permettre d'optimiser les temps d'exécution en environnement congestionné. En effet, la diminution des retransmissions coûteuses en temps sur le WAN permet d'améliorer le temps d'exécution d'une application parallèle.

Dans le but de quantifier l'impact de MPI5000, nous avons mis en place un scénario d'expérimentation permettant de créer différents contextes de congestion. Ainsi, nous pourrons répondre aux questions posées dans la problématique. Cette étude s'intéresse à des architectures avec ou sans la présence de MPI5000. Pour cela, une première étape est de mesurer les temps d'exécution d'une même application avec ou sans la présence de MPI5000. Ensuite, nous analyserons le nombre de retransmissions sur les nœuds ainsi que sur les passerelles, avec et sans la présence de MPI5000. Enfin, nous verrons l'utilisation CPU des passerelles et des nœuds avec et sans MPI5000.

Pour étudier le comportement de MPI5000 dans différents contextes de congestion, nous avons choisi d'utiliser exclusivement la fonction *MPI_Gather*. C'est une fonction de communication collective de type *Tous vers Un*. Nous allons placer le nœud destinataire sur un site géographique et les autres nœuds sur un autre site. En faisant varier ce nombre de nœud, nous allons créer une congestion plus ou moins importante sur le commutateur du site 1.

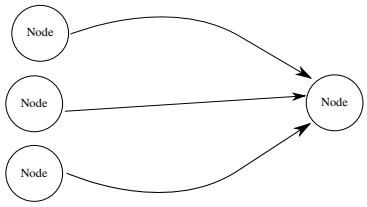


Figure 7: Description de MPI_Gather

En effet, après avoir étudié le pingpong[5], ainsi que le alltoall[6] et le scatter[7], nous avons trouvé que le gather à lui seul permet de recréer les conditions de congestion désirées.

Pour quantifier l'efficacité de MPI5000 dans les différents contextes, nous allons nous intéresser aux métriques suivantes:

- temps d'exécution de l'application : c'est le paramètre général que nous cherchons à améliorer.

[5]On mesure le temps mis par un message pour effectuer un aller-retour entre deux machines

[6]Opération de type tous-vers-tous, où des données différentes sont envoyées sur chaque processus, suivant son rang, et réarrangées suivant le rang de l'expéditeur

[7]Opération collective de type Un vers Tous

- nombre de Timeouts : c'est une indication concernant le nombre de paquets perdus.
- nombre de DupAcks : c'est une information concernant le nombre de paquets retransmis.
- utilisation CPU sur les nœuds/passerelles : indication sur le support ou non de la montée en charge.

Il existe d'autres métriques dans le cadre des flux TCP comme l'équité[8], le débit etc. que nous ne prendrons pas en compte dans notre analyse, mais qui pourraient être étudiées dans un second temps en fonction des résultats mis en évidence dans ce rapport.

10 Expérimentations

10.1 Mise en œuvre d'une expérimentation sur Grid'5000

La mise en œuvre a été effectuée sur Grid'5000 qui est une grille expérimentale de recherche. Les différents sites sont connectés par Renater. Nous n'avons réussi à faire fonctionner Web100 que sur le site de Bordeaux. Après investigation, nous n'avons pas trouvé d'explication à ce problème et nous avons été contraint d'utiliser le site de Bordeaux.

L'utilisation de la grille se fait de la manière suivante. Chaque utilisateur possède un compte Grid'5000, validé par les administrateurs. Pour pouvoir utiliser un nœud, il faut tout d'abord effectuer une réservation à l'aide des outils mis en place. L'accès à chaque site se fait par l'intermédiaire d'une machine frontale. La politique est du type premier arrivé, premier servi. Une fois un ensemble de nœuds réservés, il est possible de déployer une image personnalisée sur les nœuds. Enfin, il est possible de lancer l'application parallèle. Les contraintes d'un tel environnement d'expérimentation sont le partage du lien avec d'autres utilisateurs, la difficulté de réserver un grand nombre de nœuds. Le déploiement des images linux peuvent échouer, des opérations de maintenance sont régulièrement planifiées et provoquent des non disponibilités de certains sites. Il est globalement assez compliqué de mettre en place l'exécution d'une application parallèle sur Grid'5000.

10.2 Présentation de la plate-forme d'expérimentation

Les expérimentations sont effectués entre 2 sites : Bordeaux et Sophia. Du côté de Bordeaux, nous allons faire varier le nombre de nœuds de 2 à 19 machines. Du côté de Sophia, nous n'utiliserons qu'un seul nœud. L'architecture de la plate-forme est visible sur la figure 8. Les caractéristiques des machines sont les suivantes :

Bordeaux : AMD Opteron 2218 2.6 GHz

Sophia : AMD Opteron 246 2.0 GHz

Les nœuds à Bordeaux et Sophia sont tous connectés au commutateur avec une carte à 1Gb/s. Les commutateurs à Bordeaux et Sophia sont respectivement des HP ProCurve 5406zl et Cisco 3750.

[8] fairness