Réseau longue distance et application distribuée dans les grilles de calcul : étude et propositions pour une interaction efficace.

Ludovic Hablot

N° d'ordre: 556
N° attribué par la bibliothèque: ENSL556

THÈSE

en vue d'obtenir le grade de DOCTEUR DE L'UNIVERSITÉ DE LYON - ÉCOLE NORMALE SUPÉRIEURE DE LYONSpécialité : Informatique
LABORATOIRE DE L'INFORMATIQUE DU PARALLÉLISME
École Doctorale de Mathématiques et Informatique Fondamentale

présentée et soutenue publiquement le 17 Décembre 2009 par

Monsieur **Ludovic HABLOT**

Titre :

Réseau longue distance et application distribuée dans les grilles de calcul : étude et propositions pour une interaction efficace.

Directrice de thèse :

Madame	PASCALE VICAT-BLANC PRIMET

Après avis de :

Monsieur	CHRISTOPHE CHASSOT
Monsieur	YVES DENNEULIN

Devant la commission d'examen formée de :

Monsieur	CHRISTOPHE CÉRIN	Membre
Monsieur	CHRISTOPHE CHASSOT	Membre/Rapporteur
Monsieur	YVES DENNEULIN	Membre/Rapporteur
Monsieur	OLIVIER GLÜCK	Co-encadrant de thèse
Monsieur	RAYMOND NAMYST	Membre
Madame	PASCALE VICAT-BLANC PRIMET	Directrice de thèse

Table des matières

Table des figures

Liste des tableaux

Remerciements

Je tiens tout d'abord à remercier très sincèrement mon tuteur de thèse, Olivier, sans qui ce manuscrit ne serait pas. Ses conseils, très pédagogiques, m'ont permis de rendre mes documents beaucoup plus clairs et d'acquérir une rigueur dans la rédaction et la restitution de mes travaux de recherche.

Merci également à Jean-Christophe, pour son aide à la rédaction et toutes les discussions informelles et les anecdotes qui permettent de décompresser.

Merci à Pascale pour ses remarques scientifiques, souvent pertinentes.

Merci à tous les membres du jury, d'avoir accepté d'être rapporteurs et examinateurs de ma thèse. Toutes leurs questions soulèvent un ensemble de points problématiques auquels nous pourrons nous intéresser par la suite.

Merci à Sebastien et Romaric pour leur aide technique et morale. D'innombrables fois, ils ont su me guider pour résoudre différents problèmes. Leur soutien m'a permis de mener à bien cette thèse.

Je remercie également tous les autres *gros sacs* du laboratoire, pour leur encouragements et tous les bons moments passés ensemble. Je remercie également tous mes amis, les chanteurs et les voyageurs et les musiciens et surtout les danseurs, qui m'ont soutenu dans les moments de doutes et m'ont fait vivre des moments inoubliables en me permettant une évasion dans une autre réalité, notamment pendant la rédaction.

Je remercie enfin toutes les Vénus, déesse callipyges, sujets de nombreuses discussions passionnées au cours de ces trois années.

Résumé

Apparu en 1970, le calcul parallèle permet, contrairement aux applications classiques qui exécutent un algorithme de manière séquentielle, d'exécuter des tâches d'une même application sur plusieurs processeurs en même temps. Les premières architectures — les supercalculateurs — qui regroupaient des milliers de processeurs au sein de la même machine, ont fait place aux grappes, à la fin des années 1970 : une interconnexion d'ordinateurs standard par un réseau rapide. Ces architectures s'étant développées un peu partout, les grilles ont fait leur apparition au début des années 1990, de manière à fédérer les ressources de différentes entités en les interconnectant et ainsi disposer d'une plus grande puissance de calcul globale. La grille, telle que nous la considérons dans ce manuscrit sera donc définie comme une interconnexion de grappes par un réseau longue distance.

Les applications parallèles s'appuient la plupart du temps sur le standard MPI qui fonctionne par passage de message. Initialement destiné aux grappes, celui-ci est toujours utilisé pour programmer les communications des applications s'exécutant sur les grilles. Cela permet la réutilisation d'anciennes applications.

Alors que différents problèmes ont été résolus pour les communications au sein des grappes, le réseau longue distance de la grille pose plusieurs problèmes. Tout d'abord, les messages MPI sont transmis de manière fiable sur le réseau longue distance via le protocole TCP. Or TCP, qui reste le protocole de transport utilisé dans la plupart des grilles, est basé sur un transfert de données à l'aide de flux ; il est donc peu adapté aux communications MPI. Ensuite, la grande latence du réseau longue distance implique des communications et des retransmissions de paquets perdus qui sont coûteuses. Enfin, le débit disponible sur le lien d'accès à ce réseau est généralement inférieur à la somme des débits nécessaires si tous les processus communiquent en même temps sur ce lien. Ceci crée de la congestion à la fois au sein d'une même application et à la fois avec les autres applications qui l'utilisent, et il devient nécessaire de gérer ce goulot d'étranglement.

L'objectif principal de cette thèse est d'étudier en détail les interactions entre les applications parallèles et la couche de transport dans les réseaux longue distance des grilles de calcul, puis de proposer des solutions à ces problèmes.

Le **chapitre 2** présente un état de l'art des différentes implémentations disponibles pour l'exécution des applications MPI sur une grille de calcul. Nous étudions les efforts qui ont été fournits au niveau de la gestion de l'hétérogénéité et de la gestion de la longue distance.

Dans le **chapitre 3**, nous présentons les outils que nous avons développés pour obtenir une vue globale des communications MPI sur la grille, au niveau de chaque couche traversée. A l'aide de ces outils, nous analysons les performances des Nas Parallel Benchmarks (des applications MPI de référence) en terme de communications et de temps d'exécution. Nous avons ensuite cherché à évaluer l'environnement d'exécution, plus précisément les performances des communications sur la grille puis les implémentations MPI existantes. Dans [GHVBPS06], nous évaluons les liens haut débit de Grid'5000 (la grille de recherche française). Cette étude

a permis de mettre à jour que des optimisations étaient nécessaires afin de pouvoir bénéficier du débit maximal du lien.

A la suite de cette étude, nous avons cherché à évaluer si les applications MPI pouvaient tirer parti des capacités de communication de la grille (cf [HGM+07], [HGMVBP08]). Les résultats montrent que la grille est un environnement d'exécution valide, mais affecte particulièrement les applications qui communiquent souvent et avec de gros messages. Nous avons poursuivi cette étude afin d'étudier chacun des paramètres qui pouvaient influer sur les performances : l'implémentation MPI, la latence, la limitation du débit longue distance et le partage du réseau longue distance. Nous avons mis en évidence les fortes différences de performance obtenues avec cinq implémentations MPI (MPICH2, YAMPII, MPICH-Madeleine, OpenMPI et MPICH-G2) (cf [GHVBPS06]). GridMPI montre les meilleures performances, notamment grâce à ses opérations collectives. La latence diminue principalement les performances des applications qui communiquent souvent. Le goulot d'étranglement du WAN et le trafic concurrent affectent principalement les applications avec des communications synchrones (comme les opérations collectives).

Dans le **chapitre 4**, nous avons réalisé une étude détaillée de l'utilisation de TCP par les applications MPI, au niveau des deux mécanismes qui impactent directement les performances des applications : le contrôle de congestion et le contrôle d'erreurs. Le contrôle de congestion, réalisé à l'aide d'une fenêtre de congestion, permet de limiter le débit d'émission afin de partager le lien entre tous les flux. Différentes variantes de TCP proposent de moduler l'évolution de la fenêtre de congestion. Nous étudions tout d'abord l'impact du démarrage lent de TCP puis nous montrons que le choix de la variante de TCP pour l'exécution d'application MPI impacte les performances des applications MPI.

Le second mécanisme, le contrôle d'erreur, permet de garantir la fiabilité des transferts et de retransmettre les données perdues ou corrompues. Ce mécanisme est sensible à la latence. Nous explicitons l'impact de celui-ci à la lumière des caractéristiques des communications MPI. Comme les applications MPI attendent l'arrivée des données pour continuer leur exécution, tout retard dans la transmission implique un délai supplémentaire dans l'exécution. Dans un contexte de grille où la latence est grande, l'attente pour la détection d'une perte est lente puisqu'elle dépend du RTT, et impacte directement les applications MPI.

Afin de réduire l'impact des retransmissions et de la fenêtre de congestion, nous proposons, dans le **chapitre 5**, une couche de communication entre l'application (MPI par exemple) et le protocole de transport TCP qui s'exécute de manière automatique et transparente. Le principe général est d'introduire des passerelles à l'interface entre le réseau local et le réseau longue distance pour en différencier les communications. Ces passerelles nous permettent de mettre en œuvre le découpage des connexions TCP longue distance pour éviter au maximum les pertes et les retransmissions sur le longue distance. Ce mécanisme permet également de garder la fenêtre de congestion au plus près de la bande passante réellement disponible.

Ce travail est détaillé et évalué dans [HGM+09], et montre quelles applications peuvent bénéficier de ces optimisations. Nous précisons chaque point particulier par une analyse théorique des coûts introduits et des gains escomptés, que nous appuyons par des expérimentations. Notre analyse montre que les applications qui utilisent des opérations collectives, le surcoût généré par l'éclatement des connexions n'est généralement pas compensé par le gain sur les pertes et les retransmissions.

Abstract

Parallel applications allow to execute many tasks of the same application, on many processors in the same time. These applications were first executed on supercomputers, and then on clusters. Today, this last infrastructure is evolving towards grids. Grids, in this thesis, are defined as an interconnection of clusters by a long-distance network.

Most of MPI applications are written with the MPI standard, which describe how processes communicate, by message passing. Initially design for clusters, MPI is still used to program parallel applications in grids.

While different problems have been resolved for communications in clusters, the long-distance network of the grid raises many problems. First, MPI messages are transmitted reliably on the long-distance network via the TCP protocol. However, TCP is based on a data transfer by flows ; thus, it is not adapted to MPI communications. Secondly, the high latency of the long-distance network implied costly communications and costly retransmissions of lost packets. Finally, the available throughput on this access link to the network is generally lower than the sum of the throughput if all nodes want to communicate on this link, at the same time. This is creating congestion, both within the application and both between the other applications. It becomes necessary to manage this bottleneck.

The main objective of this thesis is to study in detail the interactions between the parallel applications and the transport layer in the long-distance network of computing grids, and then to solve these problems.

In **chapter 2**, we present the state of the art of existing implementations for executing MPI applications on a grid. We study the efforts that have been made to manage heterogeneity and long-distance connections.

In **chapter 3**, we present the tools we developped to get a global view of the MPI communications on the grid, in each crossed layer. With these tools, we analyse the performances of Nas Parallel Benchmarks (NPB, a reference in MPI applications), in terms of communications and execution time. Then, we evaluate the experimental environment, more precisely we analyse the performances of communications on the Grid. In [GHVBPS06], we measure the available throughput on high-speed links of Grid'5000 (the french research grid). This study unearth that optimizations are necessary to benefit from the maximal throughput.

Following this study, we analyze if MPI applications can take profit of communication capabilities of the grid (cf [HGM+07], [HGMVBP08]). The results show that the grid is a valid exectution environment, but affects applications that communicate often with big messages. We followed this study through to determine how each parameter can impact performances : the implementation, the latency, the low throughput of the long-distance link and the sharing of this link. Latency principally slow down applications that communicate frequently. The bottleneck and the concurrent traffic mainly affect the applications with synchronous communications (as collective operations).

In **chapter 4**, we study in detail the use of TCP by MPI applications, with regards to

the two mecanisms that directly impact the application performances : the congestion control and the error control. The congestion control is achived by a congestion window that limit the emission throughput in order to share a link between all flows. Different TCP variant propose to make this window growing differently. We show that the choice of a variant directly impact performances of MPI applications.

The second mecanism, the error control, guarantee the transfer reliability and enables to retransmit lost or corrupted data. This mecanism is sensitive to latency. We study this mecanism at the light of MPI communication characteristics. As MPI applications usually wait on data to continue their execution, each delay in transmission implied an increase of execution time. In a grid context, where latency is high, the wait for loss detection is high (depend on RTT) and directly impact MPI applications.

In order to reduce the impact of retransmissions and the impact of congestion window, we propose, in **chapter 5**, a communication layer between the application (MPI for example) and the transport protocol (TCP) that is automatically and transparently executed. The general principle is to introduce proxies at the interface between the local network and the long-distance network to differentiate communications. These proxies allows to put forward the split of TCP connections in order to avoid losses and retransmissions on the long-distance link. This mecanism also allows to keep the congestion window closer to available throughput on the long-distance network.

This work is detailled and evaluated in [HGM+09], and show which applications can benefit from these optimisations. We analyse for many points, the overhead and the benefits of the use of proxies. The theoritical analysis is supported by experiments. We conclude that for MPI applications that are using collective operations, the benefit on losses and retransmissions generally do not hide the overhead added by the splitting of the connexions. Other applications benefit from this mecanism if they communicate sufficiently.

Chapitre 1

Introduction

L'objectif général de notre thèse est l'étude de l'interaction entre le réseau longue distance et les applications distribuées dans les grilles de calcul. Dans ce chapitre, nous présentons les applications cibles et leurs architectures d'exécution, avant de préciser les hypothèses, la problématique et les contributions de la thèse.

1.1 Les applications parallèles

De nos jours, les ordinateurs sont de plus en plus utilisés pour modéliser ou simuler des problèmes scientifiques complexes. Certains calculs de physique, d'astronomie, de météorologie ou de biologie, comme par exemple ceux du CEA ou du CERN, nécessitent de pouvoir exécuter des milliards d'opérations. Le seul moyen efficace de réaliser de telles opérations est de diviser le calcul global en autant de subdivisions que possible afin de réaliser chacune d'entre elles simultanément. Apparu en 1970, le calcul parallèle permet, contrairement aux applications classiques qui exécutent un algorithme de manière séquentielle, d'exécuter des tâches d'une même application sur plusieurs processeurs en même temps. La parallélisation peut intervenir à différents niveaux : au niveau des instructions, au niveau des données, ou au niveau des tâches. Dans ce dernier domaine, OpenMP [Ope] (Open MultiProcessing), une API de programmation parallèle, est destinée à l'exécution d'applications sur des architectures multiprocesseurs à mémoire partagée. PVM [GBD+94] (Parallel Virtual Machine), un logiciel de programmation parallèle, est peu à peu abandonné au profit du standard MPI (Message Passing Interface). De ce fait, MPI [MPI94] devient le standard de fait pour l'exécution d'application MPI dans une grille de calcul.

MPI (Juin 1994, [MPI94]) (Message Passing Interface) est un standard d'une bibliothèque de communication pour applications parallèles, dont il existe plusieurs implémentations différentes, qui permet l'échange de messages entre les différentes tâches d'une même application parallèle. Il spécifie à la fois les aspects protocoles et sémantiques que doivent respecter les implémentations et définit également l'interface (c.-à-d. le prototype des fonctions) entre l'application et l'implémentation. MPI fonctionne par passage de messages tout en garantissant à la fois l'ordre des données et leur intégrité.

Le standard définit différentes fonctions point à point permettant l'échange de messages entre deux processus et des opérations collectives qui impliquent plusieurs processus. Le standard définit également une version synchrone de chacune des fonctions qui permet de garantir la transmission d'un message au destinataire avant de continuer l'exécution d'un processus. Les deux fonctions point à point de base sont MPI_Send et MPI_Recv qui permettent respectivement d'envoyer et de recevoir un message. Les opérations collectives permettent de communiquer de tous les processus vers un autre (MPI_Gather), d'un processus vers tous les autres (MPI_Bcast) ou de communiquer tous ensemble (MPI_Alltoall). D'autres fonctions comme MPI_Reduce permettent d'effectuer la même opération arithmétique sur l'ensemble des processus (addition ou multiplication).

L'utilisateur peut préciser l'architecture d'exécution par l'intermédiaire de communicateurs qui regroupent un ensemble de machines aux caractéristiques communes. Cette fonctionnalité peut, par exemple, servir à regrouper des machines accessibles par un même réseau. Seuls les membres partageant un communicateur commun peuvent communiquer entre eux mais un communicateur global contenant tous les processus permet de joindre n'importe lequel d'entre eux.

Différentes version du standard dont la plus récente date de Juillet 2008 (Version 1.3 [MPI08]), ont corrigé et apporté des précisions à la version 1.0 décrite ci-dessus. La version

MPI-2.0 du standard (Juillet 1997, [MPI97]) ajoute des fonctions de gestion des communicateurs, des fonctions de communications unidirectionnelles, des opérations collectives non-bloquantes ainsi que des fonctions d'entrée/sortie en mémoire partagée. La dernière version disponible, numérotée 2.2, date de Septembre 2009 [MPI09].

1.2 L'architecture d'exécution

L'apparition des applications parallèles ainsi que des paradigmes de communications est liée à l'évolution des architectures sur lesquelles elles sont exécutées. Cette section étudie l'évolution des architectures d'exécution en détaillant particulièrement les spécificités des grilles qui constituent la plateforme cible de notre étude.

1.2.1 Evolution de l'environnement d'exécution

Les grappes (également appelées *clusters*) sont apparues en 94 avec le démarrage du projet BeoWulf. Ces architectures interconnectent des ordinateurs "grand public", peu coûteux, par un réseau local (ou LAN — Local Area Network), dans le but de disposer d'une puissance de calcul plus grande et extensible. Différents types de réseaux sont apparus pour interconnecter ces machines : Ethernet, Myrinet [Sitc], Quadrics [Sitd] ou Infiniband [Sitb]. Leur débit peut atteindre 10 Gb/s et leur latence être inférieure à $1\mu s$. Cependant, ces performances restent légèrement en dessous de celles observées pour les communications au sein d'un supercalculateur. Il est donc plus coûteux de communiquer entre deux machines d'une grappe qu'entre deux processeurs situés au sein de la même machine.

Les grappes de calcul sont extensibles de manière à faire évoluer une grappe existante en ajoutant des machines. Comme les technologies évoluent rapidement, le type des machines constituant une grappe est rarement homogène. Il en résulte des performances différentes en terme de puissance de calcul. Ce regroupement de grappe implique également une hétérogénéité des réseaux interconnectant les machines. Pour étendre toujours plus la puissance de calcul et donc le nombre de machines de la plateforme d'exécution, il arrive qu'il soit nécessaire d'interconnecter des machines géographiquement voisines mais avec des réseaux locaux distincts, pour ne faire qu'une seule plateforme. Nous parlons alors de grappes de grappes. La figure 1.1 montre une interconnexion de deux grappes (1 et 2). La grappe 1 est connectée par un réseau Myrinet alors que la grappe 2 possède un réseau Quadrics. Les deux grappes sont interconnectées par un réseau Ethernet commun ainsi que par une passerelle qui possède les deux interfaces.

1.2.2 Les grilles

1.2.2.1 Définition

Bien qu'il existe différents types de grilles, comme les grilles de données qui permettent le stockage de données d'expérience ou les grilles dédiées à un domaine particulier permettant de mutualiser des ressources, nous nous intéressons aux grilles de calcul qui découlent tout naturellement des architectures présentées dans la section précédente. En 1999, Ian Foster et Carl Kesselman [FK99], définissent une grille de calcul de cette manière :

> *« A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. »*

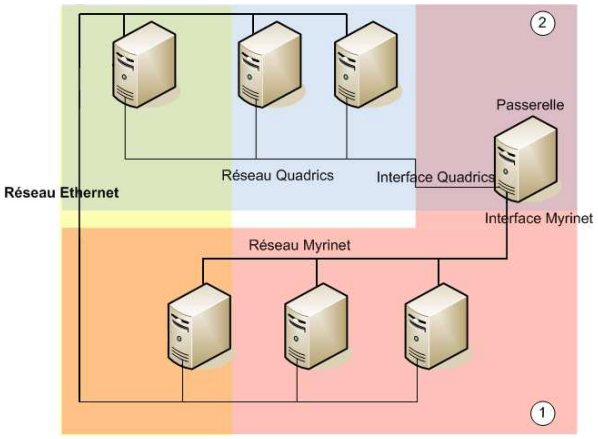


FIG. 1.1 – Exemple d'une grappe de grappes

Puis, à la suite de l'évolution des grilles, Foster propose en 2002 une nouvelle définition dans [Fos02] qui est la suivante :

> *« Coordinates resources that are not subject to centralized control using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service. »*

L'ensemble de ces deux définitions nous indique donc qu'il s'agit d'une coordination bon marché de ressources matérielles et logicielles qui fournit une haute puissance de calcul. Cette architecture propose différents services en utilisant des protocoles et des interfaces standards et ouverts. Différents projets de grille scientifiques ont vu le jour ces dernières années, parmi lesquels on peut citer EGEE [GJG+05] (Enabling Grids for E-sciencE, Europe), NAREGI [Miu06](National REsearch Grid Initiative, Japon), ou TeraGrid [C+07](États-Unis). D'autres projets comme DAS-3 [DAS](Distributed ASCI Supercomputer 3, Pays-bas), ou encore Grid5000 [BCC+06](France) sont uniquement dédiées aux recherches sur les grilles et ne servent pas de grille de production.

Pour notre part, nous utiliserons une définition plus précise mais plus réduite de la grille qui consiste à étendre encore la notion de grappes de grappes : une aggrégation de grappes ou de grappes de grappes, géographiquement éloignées et interconnectées par un réseau longue distance. Ce dernier est constitué d'un WAN (Wide Area Network) par opposition au LAN (Local Area Network). Par la suite, les termes de communications locales se réfèreront aux communications qui sont effectuées sur le LAN, tandis que les communications sur le WAN seront intitulées longue distance.

La figure 1.2 représente une vue d'ensemble de la grille Grid5000. Cette grille interconnecte 9 sites répartis sur toute la France. Chaque site possède une ou plusieurs grappes interconnectées par différents types de réseaux. L'ensemble de nos expérimentations a été effectué sur cette grille.

1.2.2.2 Spécificités des grilles

Les grilles, telles que nous les avons défini (c-.à-d. comme une aggrégation de grappes de calcul reliées par un réseau WAN dédié) présentent les caractéristiques suivantes :

1. Hétérogénéité des machines. Les machines ont des capacités de calcul (nombre/puissance des cœurs/processeurs...), des capacités mémoire et de stockage différentes. Il est donc

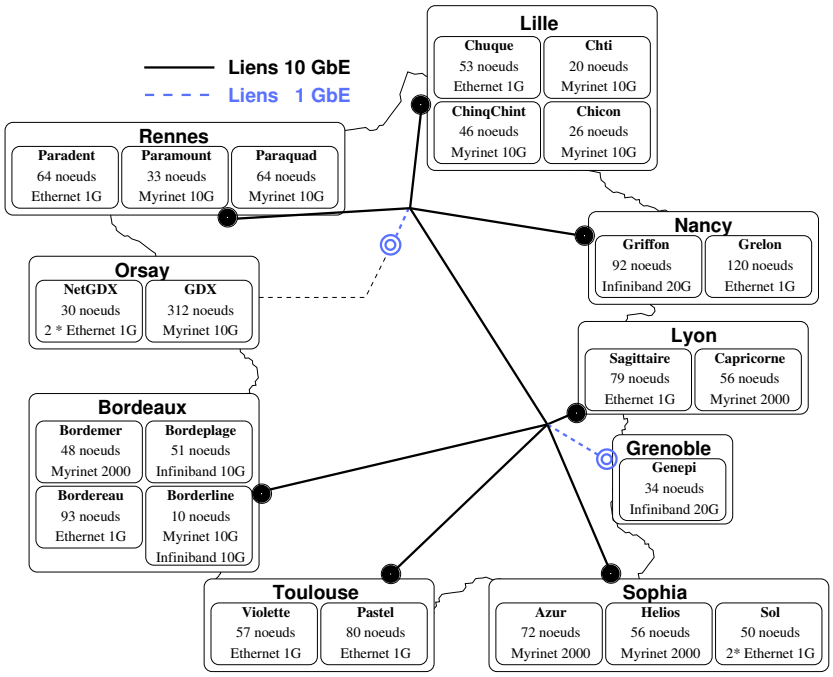


FIG. 1.2 – Interconnexion des grappes de Grid5000

nécessaire de tenir compte de cette hétérogénéité lors du placement des processus d'une application sur les resssources obtenues pour son exécution.

2. Hétérogénéité des réseaux. Chaque grappe utilise un voire plusieurs réseaux rapides permettant d'interconnecter ses noeuds localement. Ces réseaux fournissent une bande passante élevée ainsi qu'une faible latence. Ils peuvent différer d'une grappe à l'autre. Les communications locales de la grappe A peuvent par exemple utiliser un réseau de type Myrinet et celles de la grappe B un réseau de type Infiniband. La difficulté consiste alors à faire communiquer entre elles des machines qui utilisent des réseaux différents.
3. Grande latence WAN. La latence est plus grande sur le réseau longue distance (entre les sites de la grille) qu'à l'intérieur d'un même site. La latence fournie par les réseaux rapides est de l'ordre de quelques microsecondes (par exemple, $2.2\,\mu s$ pour Myrinet [1] et $\sim 1\,\mu s$ pour Infiniband [2]). De même, la latence de TCP sur Ethernet est de l'ordre de 50 μs, voire similaire à celle des réseaux rapides si l'on utilise une carte spécifique qui fournit du RDMA. A contrario, la latence sur le réseau longue distance dépend essentiellement de la distance physique entre les sites. Elle peut varier de quelques millisecondes à plus de cent millisecondes si on considère des liens intercontinentaux.
4. Goulot d'étranglement du WAN. Le débit du lien d'accès au WAN est inférieure à la somme des débits des noeuds qui peuvent communiquer dessus. La bande passante du réseau WAN est de l'ordre de 10 Gb/s alors qu'une grappe est constituée de centaines de noeuds qui peuvent communiquer sur ce réseau à 1 Gb/s. Il y a donc risque de congestion à l'interface LAN/WAN.
5. Partage des ressources. Les utilisateurs de la grille partagent ses différentes ressources : les noeuds voire les différents cœurs, leur mémoire et leur accès aux périphériques, le

[1] http ://www.myri.com/scs/performance/MX-10G/
[2] http ://www.mellanox.com/content/pages.phppg=performance_infiniband

système de stockage, les différents réseaux. Il faut donc partager et arbitrer l'accès à ces ressources.

Nous nous intéressons principalement aux questions soulevées par les trois dernières caractéristiques. Les sections suivantes précisent le cadre de nos travaux et les problèmes que nous avons traité.

1.3 Hypothèses

Notre étude porte sur l'interaction entre les applications distribuées et les réseaux longue distance dans une grille de calcul. Dans notre cas, les applications distribuées cibles sont des applications MPI. Nous évaluons cependant nos propositions pour d'autres types d'application. Comme les utilisateurs du standard MPI ne sont pas des spécialistes des réseaux et qu'ils n'ont pas forcément connaissance de l'architecture sur laquelle vont s'exécuter leurs applications, nous considérons ici qu'il est nécessaire de ne pas modifier l'application de manière à être transparent du point de vue de l'utilisateur. Nos optimisations s'intéressent donc aux niveaux inférieurs. De plus, pour tirer parti des optimisations offertes par les différentes implémentations MPI, telles que la gestion de l'hétérogénéité ou le placement des processus, nous ne proposons pas une nouvelle implémentation MPI mais nous souhaitons plutôt nous placer en dessous de celle-ci. Nos propositions sont donc totalement transparentes pour l'utilisateur et son application, et ce, quelle que soit la librairie MPI utilisée.

Les applications MPI ont pour but d'obtenir le résultat d'un problème donné le plus rapidement possible. De ce fait, la métrique principale que nous cherchons à optimiser est le temps global d'exécution ("*completion time*").

Le principal protocole de transport utilisé sur le réseau longue distance des grilles actuelles est TCP. En effet, même si d'autres protocoles ont été proposés pour les grilles, ils sont peu utilisés en pratique car il nécessitent de ré-écrire les applications qui les utilisent. De ce fait, nous supposerons que TCP est le protocole de transport utilisé sur le réseau longue distance de la grille et sur lequel nous axerons nos propositions. Nous supposerons également que nous disposons d'un moyen permettant de déterminer l'appartenance d'un noeud à un site donné, et ainsi regrouper les ressources qui sont attribuées à l'application par site.

Les ressources de la grille sont, par définition, partagées entre tous les utilisateurs de la grille. Cependant, pour garantir les performances, nous considérons une grille où les machines sont réservées et dédiées à un utilisateur pour un certain laps de temps. Ces machines disposent d'un environnement que nous pouvons maitriser de manière à pouvoir en modifier les paramètres et notamment changer le protocole de transport. Les liens réseaux, par contre, sont partagés entre tous les utilisateurs de la grille. Nous devons donc prendre en compte les différents flux que leurs applications peuvent créer. Le réseau est dédié à la grille ce qui implique que l'ensemble des flux qui traversent les liens longue distance sont produits par des applications de la grille. Nous pouvons donc éventuellement imaginer mettre en place des mécanismes pour contrôler ces flux.

1.4 Questions à résoudre

Au vu des contraintes posées par la grille et des hypothèses que nous plaçons sur l'environnement d'exécution, cette section détaille les problèmes liés à l'exécution d'applications MPI sur une grille de calcul. Notre problèmatique s'exprime ainsi :

Comment exécuter au mieux des applications MPI sur une grille de calcul dont le protocole de transport sur le réseau longue distance est TCP, en optimisant l'interaction entre ces deux couches ?

Pour étudier ce problème général, plusieurs sous-questions doivent être examinées :

Q1 – Comment se comportent les applications MPI sur un réseau longue distance ?

La première information dont nous avons besoin est d'avoir une idée précise des communications que réalise une application. En effet, le principe même du passage de messages implique qu'un programme MPI soit parfois dans l'obligation d'attendre des données avant de poursuivre son exécution. De ce fait, tout délai supplémentaire dans les communications implique un surcoût dans l'achèvement de l'exécution d'une application. Il est donc nécessaire d'avoir une idée précise des facteurs qui retardent la complétion des communications. Ces facteurs peuvent impliquer la bibliothèque de communications ou la couche de transport qu'il conviendra d'instrumenter conjointement.

Ensuite, il convient d'étudier quel est l'impact de chacune des trois caractéristiques de la grille précédement mentionnés à la section 1.2.2.2, — la latence, le goulot d'étranglement du WAN et le partage du réseau longue distance —, sur les applications MPI. Par exemple, étant liée à la distance, la latence entre les sites est incompressible. De ce fait, la grille est-elle un environnement valable pour l'exécution d'applications distribuées dont les performances en terme de temps d'exécution sont dépendantes de l'efficacité des communications ? On peut donc se demander si toutes les applications peuvent tirer parti d'une grille de calcul de manière identique. Il est nécessaire de déterminer les facteurs qui vont limiter l'exploitation d'une telle infrastructure : le nombre des communications, leur taille ou leur type ...

Q2 – Quels paramètres de TCP limitent les communications des applications MPI dans un réseau longue distance ?

TCP a été développé dans le contexte d'Internet pour réaliser un partage de bande passante dans un réseau où s'exécutent des milliers de flux. Les mécanismes de TCP tels que le contrôle de congestion et le contrôle d'erreurs permettent de limiter la quantité de données émises de manière à garantir une équité statistique entre les flux. Les caractéristiques de la grille sont très différentes de celles d'Internet tant par la nature de l'architecture d'exécution que par la nature des applications et flux qui transitent. Ainsi, dans Internet, on considère que les flux sont continus et qu'ils vont chercher à utiliser la totalité du lien disponible, ce qui n'est pas le cas avec les applications qui s'exécutent sur la grille. Sur ces dernières, la latence entre les sites rend le contrôle d'erreur et le contrôle de congestion coûteux et peu dynamiques puisque leurs algorithmes sont basés sur le temps aller-retour entre les deux parties de la connexion et contribuent à une utilisation non optimale des liens. Comme le temps d'exécution d'une application MPI dépend de la rapidité des communications, il est nécessaire de réduire les temps de transmission.

Cette latence rend également difficile l'exploitation de la totalité de la bande passante du lien longue distance. Différentes variantes de TCP ont été proposées pour résoudre ce problème et rendre le contrôle de congestion plus efficace sur un réseau longue distance à gros débit. Cependant, celles-ci, en étant plus aggressives vont engendrer plus de pertes et de transmissions. Il importe donc de trouver une variante qui offre le bon compromis entre le nombre de retransmissions et un envoi rapide sur le lien longue distance. L'identification de ces problèmes

nous amène à nous poser la question suivante :

Q3 – Comment réduire l'impact de TCP sur les communications MPI longue distance ?

Comme le contrôle de congestion de TCP ralentit l'émission des messages MPI, il convient de limiter l'impact de celui-ci. La suppression de ce mécanisme est envisageable dans notre contexte puisque le réseau est dédié et que les communications sont maitrisables. Cependant, une étude plus approfondie est nécessaire car les applications MPI elles mêmes peuvent créer la congestion à même de ralentir fortement les temps de transmission des données.

Nous n'envisageons pas de supprimer des communications au niveau MPI mais plutôt d'adapter le protocole de transport aux conditions particulières des applications MPI. Nous nous plaçons au niveau de TCP afin de faire en sorte que les transmissions de messages se déroulent le plus rapidement possible. Ceci revient à détecter au plus tôt des pertes des données MPI et donc de réduire le délai de retransmission.

Enfin, les réseaux locaux et longue distance ont des caractéristiques différentes, notamment en terme de latence et de bande passante. Il est donc nécessaire de prendre en compte le fait qu'il y ait une grille en mettant en place des mécanismes qui soient capables de différencier les deux types de réseaux, de manière à optimiser spécifiquement les communications sur le longue distance.

1.5 Objectifs et contributions de la thèse

L'objectif principal de cette thèse est d'étudier en détail les interactions entre les applications parallèles et la couche de transport (principalement TCP) dans les réseaux longue distance des grilles de calcul, puis de proposer des solutions à ces problèmes.

Le chapitre 2 présente les différentes implémentations disponibles pour exécuter une application MPI sur une grille de calcul. Il met en évidence les différentes optimisations proposées liées aux spécificités des grilles : la gestion de l'hétérogénéité et l'optimisation des communications longue distance. Nous précisons également les efforts qui ont été apportés au niveau des protocoles de transport de manière à les adapter aux caractéristiques des applications MPI.

Notre étude commence par des mesures simples de l'impact du réseau longue distance sur les applications MPI que nous détaillons dans le chapitre 3. Nous avons commencé par mettre en place des outils pour obtenir les schémas de communications des applications MPI et des informations sur les communications longue distance : le type de communications, leurs nombres et leurs tailles. Nous proposons un système qui permet d'obtenir conjointement des informations sur les appels à l'API socket depuis MPI permettant ainsi de tracer les communications de l'application sur le réseau longue distance (TCP). A ce dernier niveau, nous suivons notamment l'évolution de la fenêtre de congestion et l'évolution de l'utilisation du tampon d'émission. Nous utilisons cet outil pour analyser les communications des Nas Parallel Benchmarks [BBB+94]. Dans un second temps, nous montrons que la grille, qui permet l'augmentation du nombre de ressources attribuées à l'exécution d'un programme parallèle, peut être un environnement d'exécution valide par rapport à une grappe. Cependant, les applications qui communiquent beaucoup en un temps réduit parviennent difficilement à améliorer leur performances. Nous montrons que toutes les implémentations MPI ne sont pas équivalentes sur la grille : GridMPI est la plus efficace grâce à ses opérations collectives. Nous étudions ensuite l'impact des trois points problématiques des réseaux longue distance précédemment mentionnés qui vont limiter les performances des applications MPI (cf. Q1 section 1.4). La latence

impacte particulièrement les applications qui utilisent le plus le réseau. Le trafic concurrent qui créé de la congestion au niveau du lien longue distance, impacte également fortement le temps d'exécution des applications.

Les questions étudiées dans le chapitre 3 n'ont permis de déterminer que partiellement l'impact du réseau longue distance, nous étudions donc en détail dans le chapitre 4 l'interaction entre les mécanismes de TCP et les applications MPI (cf. Q2 section 1.4). Le contrôle de congestion, réalisé à l'aide d'une fenêtre de congestion, permet de limiter le débit d'émission afin de partager le lien entre tous les flux. Mais de ce fait, il retarde l'arrivée des messages MPI. Nous montrons que la suppression du contrôle de congestion en laissant les applications émettre au débit qu'elles souhaitent, diminue les performances d'exécution, et ce même si le lien longue distance n'est pas congestionné. A la suite de quoi, nous étudions les différentes variantes de TCP qui proposent de moduler l'évolution de la fenêtre de congestion. Les applications qui utilisent des opérations collectives sont peu impactées par un changement de variante. Cependant, certaines variante comme Illinois ou HTCP sont les plus adaptées aux programmes avec des communications de taille variées. Le second mécanisme, le contrôle d'erreur, permet de garantir la fiabilité du protocole et de retransmettre les données perdues ou corrompues. La détection d'une perte est effectuée par le récepteur et donc proportionnelle à la latence. Comme les communications MPI sont discontinues, celle-ci va intervenir tardivement et contribuer au ralentissement de l'application.

Afin de rendre plus dynamique le contrôle d'erreurs, nous proposons, dans le chapitre 5, d'éclater les connexions TCP (cf. Q3 section 1.4) de manière à différencier les différents réseaux. Nous remplaçons la connexion longue distance par trois connexions différentes : une connexion locale, une connexion longue distance, et une connexion locale. Ceci nécessite de placer des passerelles à l'interface LAN-WAN de chaque site qui nous permettent de déplacer les pertes du réseau longue distance vers le réseau local accélérant ainsi les retransmissions. Elles permettent également de garder une taille de fenêtre de congestion sur le réseau longue distance au plus proche du débit de l'application en ne conservant qu'une seule connexion sur ce lien. Sur les noeuds, la redirection vers les passerelles est réalisée par une couche à l'interface entre MPI et TCP, transparente de telle sorte qu'elle ne nécessite pas de modifications des couches inférieures et supérieures. Les résultats montrent que toutes les applications ne peuvent bénéficier de ce mécanisme car le gain généré par l'accélération des retransmissions ne parvient pas toujours à compenser le surcoût provoqué par les passerelles. Nous proposons également d'utiliser deux variantes de TCP différentes selon le type de la connexion : une variante plus équitable entre les flux pour le réseau local et une version plus aggressive sur le longue distance. Cela permet de réduire le nombre de retransmissions sur le lien local et d'améliorer l'utilisation du lien longue distance.

Chapitre 2

État de l'art

L'exécution d'applications MPI sur une grille de calcul implique essentiellement deux couches comme le montre la figure 2.1 : l'implémentation MPI s'appuie sur le protocole de transport pour effectuer ses communications.

Différents protocoles ont été proposés pour améliorer les communications dans un réseau de grille comme le détaille la section 2.1.

Quelques implémentations s'attaquent à certains des problèmes que nous avons identifiés au niveau de la couche MPI. Cependant, nous verrons qu'elles sont principalement conçues pour les grappes mais peuvent être adaptées dans le contexte des grilles si le protocole TCP est utilisé pour communiquer sur le réseau longue distance. La partie 2.2 détaille les optimisations qui ont été apportées dans ce contexte.

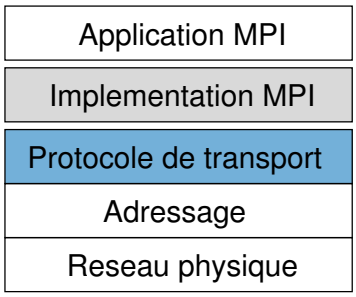


FIG. 2.1 – Les différentes couches mises en jeu par l'exécution d'une application MPI

2.1 Protocoles de transport pour les réseaux de grille

Comme nous l'avons vu dans la section 1.2.2, la grille est constituée de deux types de réseaux, le LAN et le WAN. Alors que le LAN peut-être constitué d'un ou plusieurs réseaux rapides, le WAN est constitué d'un réseau longue distance (souvent Ethernet), au dessus duquel différents protocoles de transport peuvent être employés. Dans notre contexte, il est préférable que l'interface du protocole (API : Application Programming Interface) soit proche de celle des applications MPI. Le protocole doit également être fiable c'est-à-dire contrôler les erreurs, détecter des pertes et retransmettre les données perdues. OpenMX, SCTP ou UDT, sont trois protocoles autres que TCP (qui sera étudié dans le chapitre 4) qui pourraient offrir ce genre de fonctionnalités pour une application MPI qui souhaite tirer profit du WAN.

2.1.1 OpenMX : protocole haute-performance pour cartes Ethernet génériques

MX [MX06] est un protocole de transport bas niveau à destination des réseaux rapides Myrinet. Il fonctionne par passage de message et a été développé pour être très proche des applications MPI. De ce fait, il s'attache à fournir une faible latence et une grande bande passante. C'est un protocole de transport fiable qui garantit la délivrance des messages en ordre et gére les retransmissions.

OpenMX [Gog08] est une implémentation libre de la pile MX permettant d'exécuter des applications au-dessus de cartes Ethernet classiques (ce qui n'est pas possible pour MX). Son but est de reprendre les idées de MX, afin de les adapter aux réseaux Ethernet.

OpenMX adapte ce fonctionnement La figure 2.2 montre la correspondance des différentes couches. Le driver de OpenMX se base sur la couche Ethernet classique pour transmettre

les messages. Il adapte le fonctionnement du driver MX aux cartes Ethernet pour éviter les recopies dans le driver de la carte. La latence ainsi obtenue est de l'ordre de $7\mu s$. Le contrôle de congestion est réalisé au niveau Ethernet par un système de retour de la part du switch (back-pressure) qui indique à la carte qu'elle ne doit plus envoyer de paquets.

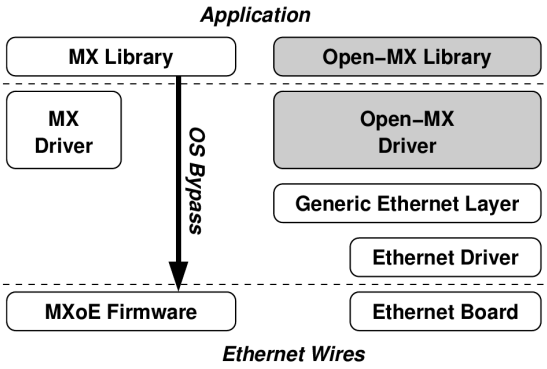


FIG. 2.2 – Comparaison des couches traversées par MX et OpenMX (de [Gog08])

Dans OpenMX, l'API de MX a été conservée ce qui permet aux applications s'exécutant au dessus de MX d'être compatibles avec l'interface de OpenMX. De ce fait, les implémentations de MPI qui prennent en compte le protocole MX dans leur bibliothèque de communications, telles que MPICH-MX, OpenMPI ou MPICH-Madeleine, peuvent tirer parti des optimisations induites par OpenMX.

Néanmoins, l'utilisation de OpenMX comme protocole de transport pour MPI dans une grille serait limitée par deux problèmes. Premièrement, l'adressage d'OpenMX est basé sur Ethernet. Or, dans la plupart des grilles, du fait de l'agrégation de ressources issues de domaines administratifs différents par différentes organisations participantes, l'adressage le plus souvent utilisé est IP. Ceci implique que le routage est également effectué au niveau IP et que les adresses Ethernet ne sont pas routées. OpenMX ne peut donc y être utilisé dans les grilles que si le routage est effectué au niveau Ethernet.

Deuxièmement, le contrôle de congestion est effectué par la couche Ethernet, plus précisément par des paquets PAUSE qui ne passent pas les équipements réseaux. Ceci implique que si le goulot d'étranglement n'est pas situé sur le premier équipement réseau traversé, toutes les files d'attentes de l'émetteur à l'équipement lent vont se remplir avant que l'émetteur ne diminue son débit. De plus, ce système implique que les deux parties, —la carte et le commutateur—, soient capables de le mettre en œuvre et soit configurées pour ça. Dans le cas où le système de back-pressure ne fonctionne pas, aucun contrôle de congestion n'est présent et cela implique une drastique perte de performances, due aux retransmissions des paquets perdus, comme nous le verrons par la suite dans la section 4.2.2.

2.1.2 SCTP : transfert de flux multimédias

SCTP (Stream Control Transmission Protocol, décrit dans la RFC4960 [Ste07]) est un protocole de transport point à point, qui garantit la transmission des données en ordre. De la même manière que plusieurs variantes de TCP sont disponibles, plusieurs implémentations de SCTP ont été développées par différentes entités principalement pour être exécutées sur différentes architectures. Comme TCP, SCTP fonctionne en mode connecté, en ce sens que les deux extrémités d'un chemin se mettent d'abord en relation avant d'envoyer les données, et

maintiennent cette relation jusqu'à ce que l'une des deux la termine. A l'inverse de TCP, SCTP transmet chaque message sur des flux différents, ce qui permet notamment de différencier la partie contrôle de la partie données.

SCTP utilise deux phases, l'une de slowstart et l'autre d'évitement de congestion. La première phase est identique à TCP. L'évitement de congestion utilise une fenêtre de congestion qui évolue de manière similaire au TCP classique sauf qu'elle se base sur la quantité de données et non sur la quantité de paquet transférés. SCTP utilise également un mécanisme d'acquittement sélectif pour indiquer plus précisément quels sont les paquets reçus.

Le fait d'utiliser plusieurs flux peut permettre de ne pas attendre sur un message en particulier et de rendre les transmissions réellement non bloquantes. L'intérêt est cependant limité aux applications qui envoient plusieurs messages à la suite pour la même destination. Il n'existe malheureusement pas de variantes de SCTP permettant de faire évoluer plus rapidement la fenêtre de congestion et SCTP souffre des même problèmes que TCP.

Les implémentations LAM-MPI [BDV94], OpenMPI [GFB+04] ou MPICH2 [Gro02] permettent d'utiliser SCTP comme protocole de transport. Kamal et Penoff [Ste07] ont utilisé LAM-MPI pour évaluer les performance de SCTP dans un LAN. Ces études montrent une amélioration des performances de MPI sur SCTP par rapport à TCP lors de l'envoi de gros messages alors que les performances avec les NAS [BBB+94] ne sont pas améliorées. D'autres expériences avec un environnement subissant des pertes de 1 à 2% montrent la supériorité de SCTP mais ce taux de pertes est bien supérieur à celui d'un WAN actuel.

2.1.3 UDT : protocole basé sur UDP pour transfert de données

UDT [GHG04] (UDP-based Data Transfer) est, comme son nom l'indique, un protocole basé sur UDP qui a pour but de transférer des flux et de partager plus efficacement la bande passante entre un petit nombre de flux réalisant de gros transferts. Contrairement aux protocoles précédents, il est développé en espace utilisateur afin d'éviter les recompilations de modules et de noyaux, mais le tampon utilisateur est fusionnée au tampon de réception, ce qui permet d'éviter l'ajout d'une recopie supplémentaire. Les acquittements sont réalisés sur la base d'une horloge (timer-based selective acknowledgement) ce qui fait que la quantité d'acquittements reçu n'est pas proportionnelle à la quantité de données envoyée. A coté de ce mécanisme, UDT utilise également un système de ACK négatif (NACK) pour signaler directement la détection d'une perte à l'émetteur. Les retransmissions sont effectuées à la fois sur la base de cette information et sur l'expiration des délais de retransmission (comme TCP).

UDT utilise deux mécanismes de contrôle de congestion. Un contrôle de débit détermine la période d'envoi des paquets et permet ainsi d'éviter les rafales en espaçant les paquets sur l'ensemble du RTT. Un contrôle par fenêtre de congestion, permet de fixer un seuil maximal de paquets non acquittés en vol, de la même manière que TCP. L'augmentation de la fenêtre suit une fonction décroissante c.-à-d. qu'elle augmente beaucoup lorsque elle est basse par rapport au seuil estimé de congestion et moins après. UDT utilise un mécanisme d'appariement de paquets (packet-pairing[1]) pour déterminer la bande passante disponible sur un lien et utilise cette information pour réaliser les diminutions de fenêtre de congestion.

Aucun version de MPI sur UDT n'est disponible, mais une intégration dans Globus [FK97] est en cours de développement et devrait permettre une exécution avec MPICH-G2 [NK03]. Le contexte des applications MPI est différent des transferts de fichiers : MPI utilise un grand nombre de connexions et communique avec des messages relativement petits comparés à la

[1] Ce mécanisme envoie deux paquets à la suite et regarde l'écart entre ceux-ci du coté du récepteur pour déterminer la congestion

	TCP	**OpenMX**	**UDT**	**SCTP**
Couche inférieure	IP	Ethernet	UDP	IP
Interface utilisateur	API socket	API MX	Propre	Propre
Niveau	noyau	module noyau	application	module noyau
Contrôle de congestion	Slowstart, fenêtre de cong.	Message du commutateur	Estimation de bdp, fenêtre de cong.	Identique à TCP
Détection des pertes	DupAck (RTT), TO(200ms+RTT)	Message du commutateur	SACK temporisé, NACK et TO	Identique à TCP

TAB. 2.1 – Comparaison de protocoles utilisables sur la grille

taille des fichiers. Cependant, le mécanisme d'acquittement négatif peut contribuer à avoir un retour plus rapide des pertes et l'utilisation du contrôle de débit en sortie permet d'éviter les rafales propres aux applications MPI. A contrario, les acquittements temporisés peuvent pénaliser les applications dont la taille des messages est faible puisqu'ils doivent attendre l'expiration du timer avant de retransmettre.

2.1.4 Comparaison des différents protocoles

Le choix d'un protocole de transport est crucial, puisqu'il va conditionner le temps d'exécution d'une application MPI et les différentes optimisations apportées au niveau de cette couche peuvent directement leur bénéficier.

Le tableau 2.1 présente une vue d'ensemble des protocoles décrits précédemment pour effectuer les communications sur le WAN. TCP est présenté à titre de comparaison. La couche inférieure précise sur quelle couche s'appuie le protocole. Le contrôle de congestion détermine et limite le débit d'émission.

OpenMX est différent des autres protocoles car il se place à un niveau inférieur. Il semble difficile à mettre en œuvre dans une grille de par son système d'adressage et son contrôle de congestion. En effet, ce dernier nécessite que tous les équipements retransmettent les informations de congestion, ce qui n'est pas le cas dans les réseaux actuels. SCTP et UDT sont plus proches de TCP. SCTP apporte en plus de TCP un mécanisme permettant de différencier chacune des communications afin d'éviter les phases de blocage. Cependant, cette optimisation semble faible et comme SCTP ne prend pas explicitement en compte les réseaux longue distance, il se comporte de la même manière que TCP. Il semble peu efficace de l'utiliser sur une grille.

UDT va plus loin en proposant des mécanismes évolués de gestion de la congestion et peut être un bon candidat pour l'exécution d'applications MPI. L'espacement des paquets tout au long du RTT est propice à réduire l'impact des rafales dues aux applications MPI. Cependant, l'utilisation UDT ou de SCTP, comme protocole de transport nécessitent cependant de réécrire les bibliothèques de communications des implémentations MPI. Il semble donc préférable d'intégrer les fonctionnalités offertes par UDT dans TCP que de réécrire les implémentations MPI.

L'analyse de la section précédente nous permet de compléter l'aperçu des protocoles disponibles pour l'exécution d'applications MPI sur la grille comme le montre la figure 2.3. La section suivante étudie les différentes optimisations qui ont été proposées au niveau de la couche supérieure c'est-à-dire au niveau de l'implémentation MPI.

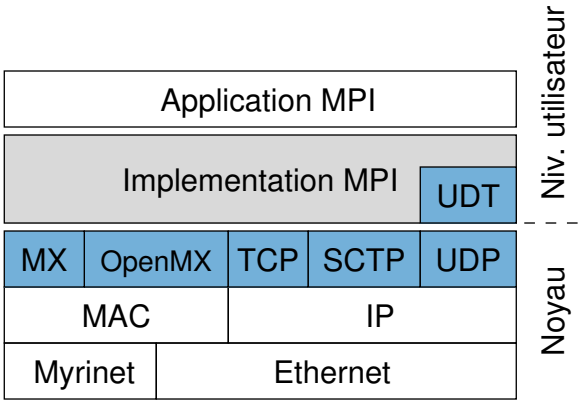


FIG. 2.3 – Les différents protocoles de transport longue distance pour les applications MPI

2.2 Implémentations MPI pour la grille

L'objectif de cette section est de présenter les différents effort qui ont été effectués au niveau de l'implémentation MPI pour l'exécution d'applications sur une grille de calcul.

Dans un premier temps, nous présentons MPICH et son architecture en couche car elle est la base de plusieurs autres implémentations. Puis nous présentons par ordre chronologique les différentes implémentations qui ont été développées. Nous nous attachons à préciser, pour chaque point de la problématique, le contexte d'utilisation et les améliorations fournies par chacune d'entre elles.

2.2.1 MPICH : l'implémentation de référence

MPICH [GLDS96] (1996) n'est pas à proprement parler une implémentation optimisée pour la grille mais elle peut être utilisée si toutes les communications se font sur TCP. Son architecture en couches peut facilement être modifiée, de telle sorte qu'elle sert de base à de nombreuses autres implémentations.

MPICH, comme la plupart des autres implémentations, utilise deux modes de transferts de données : le mode *eager* et le mode *rendez-vous*. Dans le mode *eager*, les données sont transmises directement au récepteur. A contrario, dans le mode *rendez-vous*, l'émetteur envoie seulement un message indiquant qu'il a des données à transmettre. Lorsque le récepteur reçoit l'appel à la fonction qui va réceptionner les données, il envoie un message d'acceptation à l'émetteur qui transmet les données. La différence entre les deux modes résulte en un évitement de copie sur le récepteur dans le cas de messages inattendus. La différenciation des deux modes est faite grâce à un seuil stocké en dur.

MPICH2 [Gro02] est le successeur de MPICH et y ajoute les fonctionnalités du standard MPI2.1, ainsi que des optimisations des opérations collectives à destination des clusters. Les performances de cette implémentation sont bien connus, elle servira donc de référence dans nos tests.

2.2.2 PACX-MPI : une interconnexion de MPI dédiés à des supercalculateurs

PACX-MPI [GRR99] (PArallel Computer eXtension, 1999) propose d'interconnecter deux super-calculateurs afin de les faire travailler ensemble. Les communications au sein d'une même

machine sont effectuées en utilisant une implémentation MPI dédiée, alors que les communications entre les machines sont réalisées par PACX-MPI. La décision est prise en utilisant un indicateur de niveau : le niveau local ou le niveau distant. Deux machines supplémentaires sont nécessaires afin de lancer un démon pour les communications inter-cluster. Les communications sur le WAN sont compressées, ainsi que converties pour gérer l'hétérogénéité et l'encodage des données selon le type d'architecture. PACX-MPI propose d'optimiser les algorithmes de broadcast et de reduce. Ceux-ci se basent sur les démons de manière à transférer les données une seule fois sur le WAN.

2.2.3 MagPIe : une bibliothèque d'opérations collectives optimisée pour le WAN

MagPIe [KHB+99] (2000) est une bibliothèque d'opérations collectives pour le WAN destinée à remplacer celle présente dans MPICH. De ce fait, il est possible d'intégrer MagPIe dans toute implémentation basée sur MPICH. MagPIe utilise une topologie hiérarchique à deux niveaux, le niveau LAN et le niveau WAN. Les algorithmes utilisés ont pour but d'envoyer le minimum de données sur les liens à grande latence. La plupart des opérations collectives de MPI sont optimisées et optimales pour le WAN.

2.2.4 MPICH-GQ : définition de classes de trafic

MPICH-GQ [RFG+00] (2002) a été proposé en partant du principe que le partage du réseau peut conduire à de la congestion et donc à une diminution des performances, ceci étant d'autant plus vrai si on utilise TCP comme protocole de transport. La plupart des mécanismes de QoS (Quality of Service) sont le plus souvent basés sur des flux continus (à débit constant). De ce fait, ils sont peu adaptés à des applications MPI qui communiquent par rafales.

MPICH-GQ propose de fournir au programmeur des mécanismes permettant de spécifier la classe de trafic qu'il souhaite utiliser pour les communications de son application : *besteffort*, *low-latency* (pour les messages de petite taille et les communications collectives), *premium* (pour les gros messages). Les classes de trafic sont ensuite traduites dans une couche de plus bas niveau de manière à effectuer la réservation de bande passante correspondante.

MPICH-GQ est basé sur MPICH-G2 (présenté en section 2.2.7) et sur le framework de qualité de service appelé GARA [FRS00, RS04] (General-purpose Architecture for Reservation and Allocation). La figure 2.4 propose une vue d'ensemble de l'architecture sur laquelle il repose. Le système GARA est utilisé pour réserver la bande passante et contrôler les équipements physiques à l'aide de DiffServ (Differentiated Services [NBBB98] et [BBC+98], un protocole qui permet de gérer les trafics selon différentes classes). L'agent MPI se charge de transcrire les requêtes MPI indiquant la classe de trafic en réservations équivalentes dans GARA. De manière à garantir une compatibilité avec le standard MPI (et ainsi pouvoir exécuter une application qui spécifie des classes de trafic pour MPICH-GQ sur d'autres implémentations) la spécification de classe de trafic est effectuée grâce à l'utilisation conjointe d'attribut et de communicateurs MPI. Ceux-ci seront ignorés si l'implémentation ne sait pas les gérer.

Comme les applications MPI communiquent par rafales comme précisé précédemment, le débit moyen d'une connexion n'est pas une métrique représentative. De ce fait, cela pose problème dès que l'on tente de réserver de la bande passante pour un débit donné. En effet, cela limite le débit à un niveau inférieur à celui dont l'application à réellement besoin périodiquement. Les auteurs suggèrent que le programmeur spécifie la taille maximale des messages de la classe premium de manière à réserver la bande passante en conséquence, mais aucun résultat

ne vient étayer la gestion de ce paramètre.

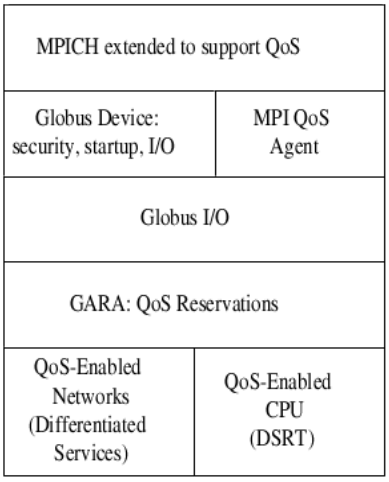


FIG. 2.4 – Architecture de MPICH-GQ (de [RFG+00])

2.2.5 MPICH-VMI : interconnexion de grappes équipées de réseaux distincts

MPICH-VMI [PJ04] (2002) permet l'exécution d'applications MPI sur la grille mais gère principalement l'hétérogénéité des grappes locales. Elle s'appuie sur la bibliothèque de communication VMI [PP02] (Virtual Machine Interface) pour effectuer ses communications entre grappes : cette implémentation utilise un mécanisme de passerelles qui permet de communiquer entre deux grappes équipées de réseaux haute-performance distincts, sans passer par TCP. Par exemple, si le cluster A est équipé d'un réseau Myrinet, le cluster B d'un réseau SCI et qu'un nœud possède les deux interfaces, des messages peuvent transiter directement de A vers B, en utilisant ce nœud comme passerelle.

MPICH-VMI construit également au démarrage une représentation de l'architecture d'exécution à deux niveaux. Cette information est utilisée pour optimiser le broadcast (comme pour les autres implémentations, en envoyant les données une seule fois sur un lien lent).

2.2.6 MetaMPICH : une architecture pour réseaux hétérogènes

MetaMPICH [PSB03] (2003) vise les architecture à réseaux hétérogènes telles que les grappes de grappes ou les grilles. Il définit trois niveaux : le niveau processeur, le niveau LAN et le niveau WAN. Pour communiquer entre deux nœuds qui n'ont pas de réseau en commun, MetaMPICH utilise des processus spécifiques pour effectuer le routage ("*routers process*" sur la figure 2.5). Un message à destination d'un réseau différent est converti de manière à pouvoir transiter en utilisant une implémentation MPI dédiée à ce réseau. L'hétérogénéité est donc effectuée grâce à une surcouche au-dessus d'implémentations MPI spécifiques.

2.2.7 MPICH-G2 : une implémentation dédiée à la grille

MPICH-G2 [NK03] (2003) est basé à la fois sur MPICH, décrit à la section 2.2.1, et le Globus Toolkit [FK97]. Globus fournit les services nécessaires à la gestion de la sécurité (gestion de

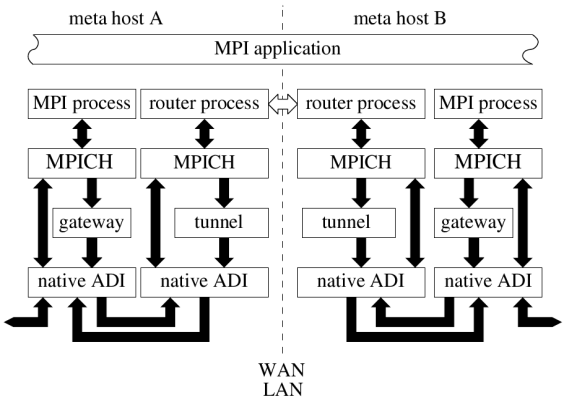


FIG. 2.5 – Architecture de MetaMPICH (de [PSB03]).

certificats), la gestion de l'exécution et la gestion des données dans la grille. MPICH-G2 s'appuie sur ses services pour allouer et gérer les ressources nécessaires à l'exécution d'applications MPI. L'hétérogénéité des réseaux est également gérée grâce à l'utilisation d'implémentations MPI dédiées. TCP est utilisé comme protocole commun entre deux réseaux. MPICH-G2 détermine le réseau le plus propice à l'envoi d'un message en utilisant une vue hiérarchique des réseaux fournie par Globus.

Dans [KSF+02], les auteurs de MPICH-G2 proposent également des opérations collectives optimisées. La hiérarchie prise en compte, basée sur la latence, est la suivante : WAN < LAN < intra-machine TCP < Vendor MPI. Si la répartition des processus sur la grille est celle de la figure 2.6 avec deux sites regroupant un supercalculateur pour le site A et une grappe de grappes pour le site B, alors l'implémentation dispose de quatre niveaux de topologie pour différencier les réseaux comme le montre la figure 2.7. Ainsi, si le processus 0 veut communiquer avec le processus 3, il utilisera un "*Vendor MPI*" puisqu'ils ont le même chiffre (0) à la profondeur 4. S'il veut communiquer avec le processus 8, il utilisera le LAN (1 à cette profondeur). Cette topologie permet d'améliorer les opérations collectives MPI_Bcast, MPI_Reduce, MPI_Barrier, MPI_Scatter et MPI_Gather en se basant sur les latences entre les processus. L'information concernant la topologie est disponible l'utilisateur s'il la requiert.

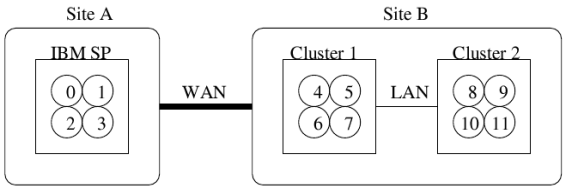


FIG. 2.6 – Vue d'ensemble de la grille pour MPICH-G2 (de [NK03]).

MPICH-G2 permet le transfert de gros messages via plusieurs flux TCP en parallèle. Le multiplexage de flux TCP a été prouvé comme un moyen efficace d'augmenter l'utilisation du lien si on utilise un TCP Reno classique [ABTV06]. Ce mécanisme est fourni par GridFTP et nécessite quelques modifications du code source de MPICH-G2.

Rank		0	1	2	3	4	5	6	7	8	9	10	11
Depth		4	4	4	4	3	3	3	3	3	3	3	3
Colors	wide area	0	0	0	0	0	0	0	0	0	0	0	0
	local area	0	0	0	0	1	1	1	1	1	1	1	1
	system area	0	0	0	0	1	1	1	1	2	2	2	2
	*v*MPI	0	0	0	0								

FIG. 2.7 – Niveaux de topologie de MPICH-G2 (de [NK03]).

2.2.8 MPICH-Madeleine : interconnexion de grappes de réseaux distincts

MPICH-Madeleine [AM03] (2003) est basée à la fois sur MPICH[GLDS96] et la bibliothèque Madeleine [Aum02]. Cette bibliothèque de communication permet l'utilisation (homogène ou hétérogène) des réseaux TCP, SCI, VIA, Myrinet MX/GM ou Quadrics. De la même manière que MPICH-VMI, MPICH-Madeleine utilise une passerelle pour communiquer entre deux grappes connectées par des réseaux différents. Les réseaux sont alors regroupés dans un même méta-réseau virtuel qui englobe tous les nœuds membres. La passerelle est un des nœuds commun au deux réseaux.

La même équipe fournit une nouvelle implémentation, MadMPI[ABFN07], basée sur la bibliothèque NewMadeleine. Cette dernière étudie la parallèlisation de la bibliothèque à l'aide de processus légers (threads) afin de faire progresser les communications plus rapidement. Elle propose également d'utiliser plusieurs réseaux rapides s'ils sont disponibles.

2.2.9 GridMPI : une implémentation dédiée à la grille

GridMPI [Gri] (2004) est une implémentation spécifiquement développée pour les grilles. Elle est composée de deux parties principales distinctes représentées sur la figure 2.8 : l'une dédiée aux communications nommée YAMPII, l'autre est l'implémentation du protocole IMPI [GHD00] (Interoperable MPI) pour gérer l'interopérabilité de différentes implémentations. IMPI permet à plusieurs implémentations MPI de communiquer ensemble en ayant un protocole commun. Comme IMPI est une surcouche à MPI, il ne modifie pas le fonctionnement de la librairie MPI. Chaque grappe utilise une implémentation MPI spécifique, propriétaire pour les communications locales sur réseau rapide (noté VendorMPI sur la figure) et communique avec un autre grappe via IMPI en utilisant une passerelle par site.

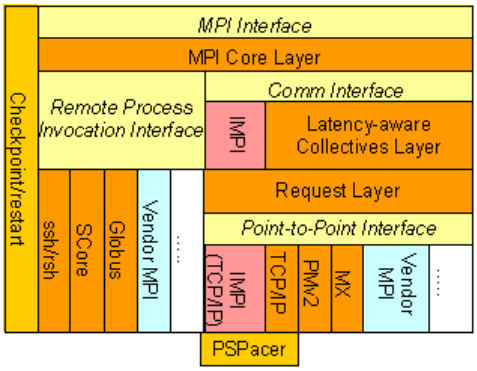


FIG. 2.8 – Architecture de GridMPI (de [Gri]).

GridMPI optimise les opérations collectives MPI_Bcast and MPI_Allreduce (cf [MKK+06]). La fonction de *broadcast* modifie l'algorithme de Van De Geijn [BGP+94] : il découpe les mes-

sages et envoie chaque partie simultanément sur le réseau de manière à utiliser au mieux ses capacités. Un message est donc découpé et distribué localement avant d'être envoyés par N nœuds sur le WAN. N est déterminé en fonction de la bande passante disponible entre les grappes. Le *allreduce*, quand à lui, modifie l'algorithme de Rabenseifner [Rab04] et effectue un reduce local avant d'échanger les données entre les clusters. De la même manière que précédemment, le nombre de nœuds qui émettent sur le WAN est fonction de la bande passante.

Dans l'article [MKK+05], les auteurs de GridMPI modifient et évaluent la couche TCP pour l'adapter aux contraintes de la longue distance et des applications MPI :

- ajout d'un mécanisme d'espacement de paquets lors de la phase de démarrage lent de TCP (cf 4.2.1) pour atténuer les effets de rafales dues aux communications MPI. Ce mécanisme espace les paquets en insérant, au niveau Ethernet, des paquets PAUSE (des paquets de type spécifique) qui seront éjectés par le commutateur. De cette manière, une communication MPI ne génère plus de rafale de paquets. Ceci est notamment intéressant pour les opérations collectives qui rendent propices la création de congestion dans les commutateurs. Lors de la phase de démarrage lent, une perte de paquet est beaucoup plus coûteuse que lors de la phase stable parce qu'elle va engendrer un passage en phase stable où la fenêtre grandit linéairement et non de manière exponentielle.
- réduction des temps de retransmission (RTO) de TCP afin de retransmettre les paquets perdus plus rapidement. La hauteur de cette réduction est basée sur la latence maximale entre deux sites de la grille.
- utilisation de deux fenêtres de congestion différentes pour les opérations collectives et les communications point-à-point. En effet, comme les opérations collectives créent plus facilement de la congestion, celle-ci va engendrer des pertes qui vont diminuer la fenêtre. De ce fait, l'émission de paquets des communications point à point qui suivra sera limitée par taille de fenêtre de congestion inadaptée au conditions du réseau.
- contournement de la couche socket pour éviter une recopie en espace noyau, à l'aide d'un driver spécifique. Les messages sont donc copiés directement de la carte réseaux vers l'espace utilisateur.

De plus, GridMPI, propose des passerelles permettant de communiquer entre des réseaux privés [TMK+08]. Si le débit de sortie d'une seule passerelle est inférieur au débit du lien longue-distance, plusieurs passerelles sont multiplexées afin de mieux utiliser la bande passante disponible sur le WAN.

Les auteurs de GridMPI proposent également de limiter la bande passante utilisée par chaque processus, en utilisant un mécanisme de d'espacement des paquets tel que décrit précédemment (cf [TMK+07]). Celui-ci est opéré par PsPacer [TKK+05] qui peut contrôler précisément le débit des données envoyées sur le WAN, toujours dans le but d'éviter la congestion et les retransmissions coûteuses. Cette limitation est basée sur une variable par processus, le MATB (Maximum allowable transmission bandwidth), qui détermine la bande passante maximale à laquelle peut envoyer un processus. Pour les opérations collectives, le MATB peut être facilement calculé puisqu'il dépend uniquement de l'algorithme utilisé. Celui-ci est calculé en divisant la bande passante du WAN par le nombre de processus qui communiquent dessus. Par exemple, si seulement la moitié des N nœuds utilisés communiquent sur le WAN de bande passante B, le MATB est égal à $B/(N/2)$. Dans ce cas, GridMPI intègre directement le calcul et l'utilisation du MATB dans l'implémentation. Pour les communications point à point par contre, il faut se baser sur le schéma de communication de l'application. C'est donc le programmeur qui doit préciser le MATB. B un paramètre de l'environnement spécifié soit par le programmeur soit par l'administrateur de la grille. A l'aide de ce paramètre et du nombre de nœuds N utilisés par son application, le programmeur spécifie le MATB à appliquer aux

communications.

Les résultats basés sur les NPB (cf section 3.1.4.1) montrent la validité de l'approche avec une légère réduction des temps d'exécution si l'on utilise la limitation de débit. Les auteurs notent même que pour certaines applications il est préférable de limiter le débit à un niveau inférieur à B/N.

2.2.10 OpenMPI : une implémentation de grille, ouverte et modulable

OpenMPI [GFB+04] (2004, souvent abrégé OMPI) est une implémentation libre du standard MPI-2 reconstruite de zéro en combinant les points forts et les ressources de projets précédents (FT-MPI, LA-MPI, LAM/MPI et PACX-MPI) ainsi que de nouveaux concepts. Il s'agit d'une des implémentations les plus utilisées actuellement. C'est une combinaison d'ORTE (pour Open Run Time Environment), un environnement d'exécution, et de la librairie MPI en elle-même.

La librairie OpenMPI est basée sur une architecture à base de composants qui permet aux développeurs de pouvoir ajouter facilement des modules. De ce fait, c'est un projet très dynamique, où il est difficile de savoir ce qui a été intégré et ce qui reste de la preuve de concept. La progression des communications est gérée par un module appelé PML pour Point-to-point Management Layer (couche de gestion point-à-point) que l'on peut voir sur la figure 2.9. Ce module peut utiliser différents composants de transfert des données (BTL pour Byte Tranfer Layer), une pour chaque protocole, fournissant ainsi un support pour l'hétérogénéité des réseaux. Les protocoles actuellement supportés sont TCP, Myrinet MX/GM et Infiniband OpenIB/mVAPI.

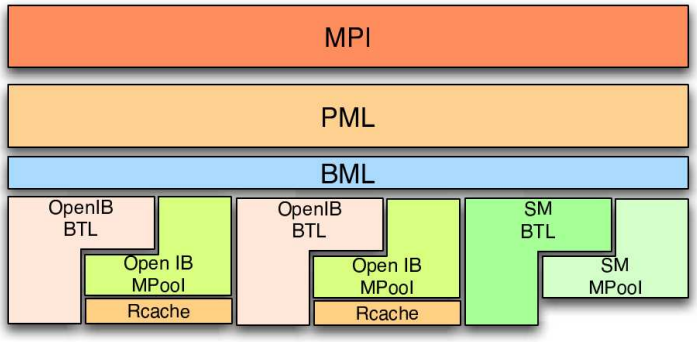


FIG. 2.9 – Architecture des différents composants de communication dans OpenMPI (de [GFB+04]).

2.3 Synthèse

Comme nous l'avons précisé dans la section 1.2.2.2, les grilles apportent les nouveaux problèmes suivants qu'il est nécessaire de gérer : l'hétérogénéité des nœuds, l'hétérogénéité des réseaux rapides des grappes, la forte latence des liens longue distance, le goulot d'étranglement du WAN et le partage du réseau par différents utilisateurs.

Le premier point peut être résolu en optimisant le placement et l'ordonnancement des tâches dans une grille de calcul. De précédents travaux ont pris en compte les caractéristiques des communications pour réaliser un placement efficace des processus en fonction des ressources : soit par une analyse post-mortem comme dans [GS03] et [BLM04] ou d'une analyse

à la compilation dans [SJM06]. D'autres travaux [CHC09] laissent la description de la topologie de l'application au programmeur et effectuent ensuite l'appariement avec la topologie physique. Aussi, même si les caractéristiques des réseaux peuvent influer sur le placement de tâches sur la grille, ce point ne sera pas traité.

Les autres problèmes ont en partie été traités par les implémentations présentées dans la section précédente. Le tableau 2.2 résume l'ensemble des optimisations disponibles pour exécuter des applications MPI sur la grille. Trois groupes se distinguent selon le point problématique étudié : la gestion de l'hétérogénéité des réseaux rapides locaux, l'optimisation des opérations collectives sur le réseau longue distance et les efforts apportés à TCP pour l'optimisation des communications longue distance. La connaissance de la topologie a été ajoutée car elle est nécessaire pour connaître les réseaux disponibles pour communiquer entre deux noeuds (réseau rapide, LAN ou WAN). Le tableau présente également les dernières publications pour avoir une idée de l'activité du projet.

La plupart du temps, la description de la topologie est effectuée de manière manuelle par l'utilisateur (MagPIe, PACX-MPI, MetaMPICH, ou MPICH-Madeleine). Dans MPICH-VMI ou OpenMPI, la découverte des réseaux rapides est effectuée au démarrage. GridMPI se base sur un variable d'environnement qui donne la bande passante entre les sites. Enfin, dans MPICH-G2, la topologie est gérée par un service de Globus qui contient la description globale de l'architecture. Selon les implémentations, la vue hiérarchique ainsi créée contient donc plusieurs niveaux d'aggrégation : même processeur, même machine, même réseau local et même réseau longue distance.

La gestion de l'hétérogénéité permet d'utiliser les réseaux rapides disponibles au sein des grappes pour communiquer localement. Les mécanismes à mettre en œuvre doivent permettre de communiquer sur ce type de réseaux (Myrinet, Infiniband ou Quadrics) en plus d'Ethernet. L'hétérogénéité peut être gérée de deux manières différentes : soit au niveau de la couche de communication qui effectue la retransmission d'un réseau à l'autre (OpenMPI, MPICH-VMI ou MPICH-Madeleine), soit au niveau MPI en créant une surcouche qui retransmet les messages à une implémentation de MPI spécifique pour ce type de réseau (PACX-MPI, MetaMPICH, MPICH-G2 ou GridMPI).

Connaître la topologie de la grille, à minima savoir si une communication va traverser le réseau longue distance, est important pour limiter au strict minimum les communications sur le WAN. A l'aide de ce paramètre, PACX-MPI, MagPIe, MPICH-VMI, MPICH-G2 ou GridMPI proposent des algorithmes optimisés pour une partie variable des opérations collectives. La topologie permet également de proposer d'autre optimisations pour améliorer les communications MPI au niveau du protocole de transport du WAN (TCP). GridMPI propose plusieurs optimisations à ce niveau en modifiant le comportement de TCP pour l'adapter à MPI. GridMPI propose de diminuer le délai de détection d'une perte (RTO), d'utiliser deux fenêtre de congestion différentes selon le type de communications (opération collectives ou point-à-point) et de limiter le débit d'émission lors de la phase de démarrage lent de manière à éviter la congestion. Il propose également un système à base de passerelles permettant de communiquer entre deux réseaux privés. MPICH-G2 s'adapte au comportement de TCP en augmentant le multiplexage sur le WAN. Ceci permet de mieux utiliser la capacité du lien longue distance pour l'envoi de gros messages. GridMPI et MPICH-GQ proposent de limiter le débit des connexions TCP relatives aux applications MPI de manière à éviter la congestion sur le WAN. Cependant, si cette limitation est réalisable facilement pour les opérations collectives dont on connaît à priori le schéma de communication, elle implique de s'en remettre au programmeur pour toutes les communications point-à-point.

La figure 2.10 reprend les points problématiques évoqués précédemment et présente les

couches où des optimisations ont été proposées pour les communications MPI sur la grille. La partie implémentation MPI est représentée en gris. La gestion de l'hétérogénéité intervient soit dans la couche (1) si l'implémentation utilise une surcouche soit dans la couche (2) si l'hétérogénéité est gérée au niveau de la bibliothèque de communication. L'optimisation des opérations collectives est effectuée dans la couche correspondante. Les optimisations de la couche de transport sont effectués dans la bibliothèque de communication ou directement dans le protocole de transport.

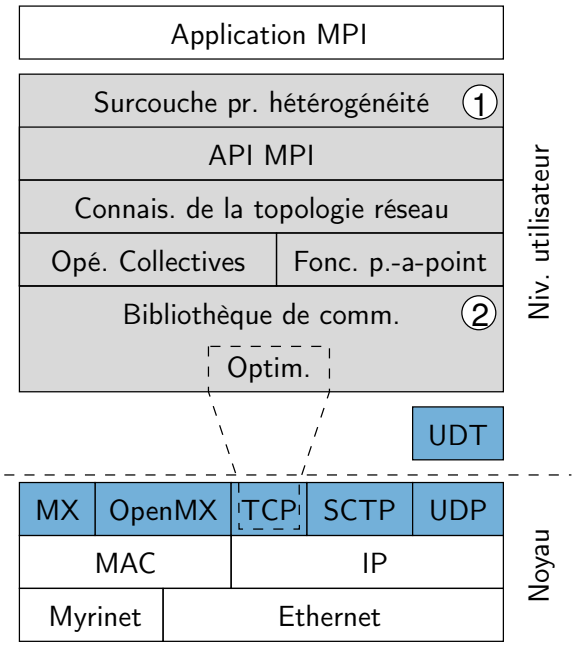


FIG. 2.10 – Interactions des différentes couches lors de l'exécution d'une application MPI

Conclusion

Dans le cadre de l'exécution d'applications MPI sur une grille, les communications locales au sein d'une grappe (ou grappe de grappes) sont généralement optimisées dans les implémentations citées précédement. Les communications au sein de réseaux hétérogènes d'un même site sont généralement gérées efficacement. La plupart des implémentations prennent en compte la topologie réseau de l'architecture d'exécution de manière à rendre optimales les opérations collectives : en minimisant les communications longue distance qu'elles induisent. En revanche, à l'exception de GridMPI et très légèrement MPICH-G2, aucune implémentation ne s'est réellement attaquée au problème de l'optimisation des communications MPI sur la longue distance.

TCP semble toujours être le protocole de transport le plus utilisé pour réaliser les communications entre les différents sites de la grille. Par contre, peu d'études ont regardé en détail l'interaction entre TCP et les applications MPI pour la grille. GridMPI est le projet le plus avancé dans ce domaine mais la plupart de leurs solutions impliquent des modifications du noyau, difficiles à maintenir à chaque évolution du noyau et à mettre en œuvre puisque l'utilisateur n'a pas forcément la possibilité de modifier le noyau des machines de la grille. Leurs propositions n'étudient pas les limitations de TCP liées au contrôle de congestion et peu celles liées au contrôle de fiabilité. Nous étudions ces problèmes dans le chapitre 4.

La limitation de débit sur le réseau longue distance proposée par GridMPI ou MPICH-GQ permet de prendre en compte le goulot d'étranglement qui peut se créer à l'interface entre le réseau local et le réseau longue distance. Ainsi chaque connexion longue distance limite le débit d'émission sur ce réseau. Cependant, ces solutions impliquent que le programmeur connaisse les communications qui ont lieu sur le longue distance. Nous proposons et évaluons dans le chapitre 5 une architecture à base de passerelles qui permet d'éclater les connexions de TCP pour toutes les applications sans en connaître les communications à priori.

	Gestion de l'hétérogénéité des réseaux	Connaissance de la topologie réseau	Optimisation des comm. longue distance		Première/ Dernière publi.
			Opérations coll.	Optimisation TCP	
PACX-MPI	Surcouche au-dessus des MPI propriétaires	Manuelle, 2 niv.	Optimisation de Bcast et Reduce	Aucune	1997 / 2003 Intégré à OpenMPI
MagPIe	Aucune	?	Optimisations de toutes les opérations	Aucune	1999
MPICH-GQ	Aucune	Aucune	Aucune	Lim. débit	2000
MPICH2	Aucune	Aucune	Aucune		2002 / 2006
MPICH-VMI	Passerelle entre réseaux rapides Support de TCP, Myrinet GM Infiniband VAPI/OpenIB/IBAL	Automatique au démarrage, 2 niv.	Optim. de Bcast	Aucune	2002 / 2004
MetaMPICH	Surcouche au-dessus des MPI MPI propriétaires	Manuelle, 3 niv.	Aucune	Aucune	2003
MPICH-G2	Surcouche au-dessus des MPI propriétaires	Service de Globus, 4 niveaux Accessible à l'util.	Optim. de Bcast, Reduce, Barrier, Gather, Scatter	Flux parallèles pour les gros messages sur le WAN	2003, mais Globus m.-à-j. régulièrement
MPICH-Madeleine	Passerelle entre réseaux rapides Support de TCP, SCI, VIA, Myrinet MX/GM, Quadrics	Aucune	Aucune	Aucune	2003 / 2007 remplacé par Mad-MPI en 2008
GridMPI	Surcouche au-dessus des MPI propriétaires via IMPI	Manuelle, 2 niv. basée sur la bande passante	Bcast et All_reduce	Dim. RTO, lim. débit pacing au démarrage, chgt. fenêtre cong., passerelles entre réseaux privés	2004 / 2008
OpenMPI	Passerelle entre réseaux rapides Support de TCP, Myrinet MX/GM, Infiniband OpenIB/mVAPI	?	?	Aucune	2004 / 2009

TAB. 2.2 – Comparaison des caractéristiques des implémentations MPI pour la grille.

Chapitre 3

Méthode et outils pour l'analyse des performances d'exécution des applications sur la grille

Introduction

Les applications MPI proviennent de différents domaines et effectuent des communications variées, vu que le standard MPI décrit uniquement des fonctionnalités et ne limite pas leur utilisation. Aussi, comme les performances de ces applications sont liées à la rapidité des communications, il est nécessaire de savoir comment elles communiquent pour pouvoir expliciter leurs temps d'exécutions sur la grille. Des travaux existants [BH09], se basant sur un modèle pLogP [KBV00], tentent de masquer, grâce à des métriques simples, la variabilité et la complexité du réseau d'interconnexion. Cependant, cette approche reste limitée et difficile à généraliser. La raison principale vient de la complexité du système sous-jacent et en particulier de TCP qui s'adapte automatiquement aux surcharges de trafic qu'elles soient internes ou externes à l'application et que les modèles de trop haut niveau ont du mal à capturer. Ainsi, nous devons être en mesure de déterminer les paramètres qui retardent l'envoi et la réception de messages MPI sur le réseau longue distance, au niveau de chaque couche traversée.

Ces paramètres doivent nous permettre de répondre à la première question que nous avons évoqué : quelles sont les performances d'exécution des applications MPI sur un réseau longue distance (cf. Q1 section 1.4) ?

Dans un premier temps, nous justifions les paramètres que nous devons capturer et analyser pour obtenir une vue d'ensemble des communications de manière à en comprendre leurs temps d'exécution. Nous proposons ensuite InstrAppli, un système qui permet d'obtenir conjointement des informations sur les appels à l'API socket depuis MPI et sur l'utilisation des connexions TCP. Ces outils nous permettent de tracer les communications de l'application sur le réseau longue distance et d'en analyser les communications. Nous utilisons cet outil sur les NAS Parallel Benchmark [BBB+94] afin de comprendre comment ils communiquent.

Dans un second temps, à l'aide de ces informations, nous étudions les temps d'exécution des applications MPI dans une grille de calcul. Nous commençons par une analyse des performances du réseau longue distance de notre grille afin d'être sûrs de ses capacités. Ensuite, nous cherchons à déterminer si la grille, en augmentant les ressources mises à disposition, peut être un environnement valable pour l'exécution d'applications MPI en comparaison à une grappe. Enfin, nous étudions comment chacune des trois caractéristiques de la grille précédement mentionnés à la section 1.2.2.2, — la latence, le goulot d'étranglement du WAN et le partage du réseau longue distance —, va limiter l'efficacité d'un passage sur la grille.

3.1 Analyse des communications des applications MPI

3.1.1 Méthodologie

Lors de nos premières expériences d'exécution d'applications MPI sur la grille, nous avons naturellement constaté des pertes de performances importantes en comparaison à l'exécution sur une grappe avec autant de ressources. La différence entre les grilles et les grappes se situe au niveau du réseau qui interconnecte les sites de la grille. La diminution de performances affecte donc uniquement les communications. Nous avons donc cherché à comprendre comment la traversée des différentes couches de communication pouvait ralentir l'exécution d'une application.

Les applications MPI font appel aux fonctions de l'API MPI, qui appellent elles-mêmes les fonctions de l'API sockets. Ces appels se traduisent par l'envoi ou la réception de segments TCP, puis de trames Ethernet sur la couche physique. Nous ne maitrisons ni l'application – écrite par le programmeur – ni les couches Ethernet et physiques – où les optimisations sont

très difficiles à mettre en oeuvre, nous allons donc nous intéresser aux couches TCP et MPI. Les données suivantes sont nécessaires pour identifier les points qui limitent les communications :

- le nombre et la taille des messages impliqués nous indiquent le volume de données échangées sur une connexion entre deux processus ;
- le schéma des communications des applications nous donne une vue d'ensemble des processus communiquants ; il est obtenu à l'aide du placement des processus sur les différents noeuds et de la répartition des noeuds sur les sites ;
- la date des données précédentes nous indique l'évolution temporelle des communications au sein d'une même application (communications en rafales ou régulières). Cette date nous permet également de faire la correspondance entre les événements de la couche TCP et les `write`.
- la fenêtre de congestion et le nombre de retransmissions sont deux facteurs qui peuvent limiter l'envoi des messages sur TCP : l'un en limitant la quantité de données envoyée sur une connexion, l'autre en retardant l'achèvement d'un envoi ;

Nous avons opté pour une analyse à posteriori à l'aide des informations récoltées par une première exécution de l'application sur un environnement réel. En effet, même si il est possible d'obtenir les communications d'une application sans exécuter le code, il est impossible de connaitre les temps de traversée des piles TCP et quels paramètres peuvent influer. Il nous était également difficile d'avoir recours à un simulateur. Bien qu'il existe des simulateurs au niveau MPI tels que MPI-Sim [PB98] et des simulateurs au niveau TCP comme NS-2 [NS2], ceux-ci ne permettaient pas d'obtenir d'informations conjointes c'est-à-dire de savoir quels événements de la couche MPI engendraient des événements au niveau de la couche TCP.

3.1.2 Outils existants de traçage et de visualisation des communications

Cette section présente différents outils qui permettent de récupérer partiellement les informations identifiées dans la section précédente. Leur caractéristiques détaillées sont présentées dans l'annexe B Généralement, les outils s'intéressent à deux aspects : soit obtenir le profil d'une application, c.-à-d. une vue résumée de ses caractéristiques, soit obtenir le traçage de l'enchaînement des événements au cours du temps. Les informations qui nous intéressent concernent les deux aspects. L'outil recherché est donc hybride, basé sur le profilage des communications mais permettant d'obtenir également une trace des événements au cours du temps.

Le choix d'un outil va dépendre de la couche dans laquelle est située l'information recherchée.

Au niveau MPI, TAU [SM06] ou Paje [CdKdOS00] permettent d'obtenir la trace des communications mais ne proposent pas d'obtenir le schéma global des communications effectuées ni le regroupement par site. La version 3D de Paje, proposée par Schnorr [SHON08], visualise les communications par une représentation spatiale et temporelle : le plan représente la localisation des processus, la troisième dimension les communications au cours du temps. Le regroupement des processus par site est effectué grâce à la localisation physique de noeuds de la grille qui est fournie à l'application. Cependant, elle ne permet pas d'agréger les informations obtenues par connexion.

Du coté des communications TCP, KTAU [NMSM06] enregistre les appels à différentes fonctions du noyau Linux mais ne permet pas de visualiser l'évolution des différentes variables de TCP, comme par exemple, la fenêtre de congestion. Web100 [MHR03], par contre, fournit l'ensemble des informations intéressantes mais ne fournit pas l'évolution de ces valeurs au cours du temps. Il est donc nécessaire de relever ces valeurs périodiquement mais cela en plus d'impacter les performances du système, ne nous informe pas précisément à quel moment elles

ont eu lieu. Enfin, tcpprobe consigne effectivement les événements sur une connexion TCP au cours du temps lors de l'arrivée de paquets (données ou ACK) mais il n'est pas suffisant en l'état vu qu'il ne nous fournit que l'évolution de la fenêtre de congestion, et des numéros de séquence des paquets TCP.

Malgré leurs limitations, KTAU et TAU offrent une idée intéressante qui est d'offrir une visualisation conjointe des événements des deux couches en synchronisant leurs événements à l'aide d'une horloge matérielle. L'idéal serait donc un outil, qui, à la manière de TAU et KTAU, récupérerait les informations aux deux niveaux mais en ajoutant les informations qu'ils ne fournissent pas. Cet outil doit permettre, à la fois d'être plus proche de TCP afin d'obtenir des informations sur les événements qui limitent les émissions (l'évolution de la fenêtre de congestion et le nombre de retransmissions) et, à la fois, d'agréger les informations sur les communications MPI (nombre et taille des messages) en les regroupant par connexion et par site. La section suivante détaille InstrAppli, l'outil que nous avons développé de manière à prendre en compte tous ces paramètres.

3.1.3 Récupération transparente des informations sur les communications aux niveaux MPI et TCP

Cette section détaille l'utilisation conjointe de deux outils qui permet d'obtenir les informations sur les communications longues distances d'une application MPI au niveau des couches MPI et TCP. La première partie est basée sur une librairie qui surcharge les fonctions de la couche socket de manière à enregistrer les appels à ses fonctions et leurs paramètres. La seconde partie est une adaptation du module tcp_probe de manière à obtenir l'évolution de la fenêtre de congestion et les retransmissions effectuées au cours du temps.

Les informations obtenues par les deux outils sont synchronisées dans le temps grâce à l'horloge système du noeud. La synchronisation entre les différents noeuds n'est pas nécessaire étant donné la grande latence entre les noeuds. L'appariement des échanges au niveau MPI avec ceux de TCP est effectué par l'utilisation des ports et des adresses IP des deux extremités de la connexion.

3.1.3.1 La librairie InstrAppli : surcharge des fonctions de l'API socket

La librairie InstrAppli permet de tracer l'ensemble des communications d'une application MPI lors des appels à l'API socket. Le principe est de détourner les appels systèmes aux fonctions de l'API socket de la librairie standard de Linux, en chargeant notre librairie en premier. Ce système peu instrusif et transparent, permet d'être en-dessous de l'implémentation MPI et d'être adaptable à chacune d'entre elles. De ce fait, InstrAppli ne permet pas de consigner les appels aux fonctions MPI telles que MPI_Alltoall ou MPI_Send mais si l'on cherche à optimiser l'interaction entre MPI et TCP, ce niveau est le plus approprié.

Le détournement s'effectue sous linux grâce à la variable LD_PRELOAD qui permet de charger une librairie —InstrAppli dans notre cas— avant les librairies avec lesquelles un programme a été lié lors de la compilation. InstrAppli va être initialisée lors du premier appel à la librairie standard. Elle redéfinit les fonctions *read, readv, recv, recvfrom, recvmesg, write, writev, send, sendto, sendmsg* afin d'effectuer le traitement nécessaire au stockage des communications. Il suffit ensuite d'appeler la fonction originale pour que l'application puisse s'exécuter de manière transparente. Les éléments consignés sont la date système, le port source et le port destination, les adresse IP sources et destination, la fonction appelée et la taille des données impliquées. La date système est obtenue grâce à la fonction *gettimeofday* qui est suffisamment

précise pour conserver la précédance des événements.

Chaque noeud est identifié en fonction de son adresse IP. Cette information nous permet de regrouper les noeuds par site : soit parce que la plage d'adresse est différente sur chaque site (comme c'est le cas dans notre grille), soit parce qu'un service nous fournit la correspondance entre une adresse et son site. La différenciation entre les communications locales et longue distance est donc réalisée à ce niveau.

Pour réaliser le schéma de communication, on ne conserve que les fonctions qui écrivent sur la socket. Les connexions où il y a très peu de communications sont supprimées, c'est à dire celles avec un nombre de *write* inférieur à dix et dont la quantité de données n'excède pas 1 Kio [1]. Ces communications sont considérées comme des messages de contrôles et gênent la vision globale que l'on souhaite obtenir.

Enfin, les données obtenues sont tracées avec GraphViz [GKNpV93] qui permet de tracer des graphs, et ce de manière automatisable. Dans notre cas, il nous permet d'obtenir une vue d'ensemble des communications d'une application avec un noeud par machine, regroupés par grappe. Les boites représentent les noeuds identifiés par leurs adresses IP, les arêtes orientées représentent les communications. Les chiffres sur les arêtes affichent le nombre de `write` et la quantité de données transférée sur cette connexion. Les communications sont différenciées selon si elles utilisent ou non le lien longue distance (respectivement en rouge et en bleu).

La figure 3.1 montre le schéma de communication de CG (une application MPI des NAS Parallel Benchmark décrits plus loin) et les informations aggrégées sur les connexions. On constate que seule la moitié des noeuds de chaque site communique sur le réseau longue distance. Les communications sont effectuées à l'aide de 1978 messsages par noeud dont la somme représente 282 Mio.

Afin de compléter cette vue, `InstrAppli` trace également un graphique des `write` au cours du temps, avec en ordonnée la taille des données écrites. La figure 3.2(a) est un exemple des `write` sur une des connexions longue distance lors de l'exécution de CG. On constate qu'il communique constamment avec des messages de 150 Kio.

3.1.3.2 Adaptation du module `tcp_probe` pour l'instrumentation de la pile TCP

L'outil que nous avons présenté dans la section précédente opère au niveau de l'API socket, et fournit l'ensemble des événements qui se produisent sur une socket. Ces informations doivent être associées aux événements de la couche TCP pour déterminer les facteurs limitants lors de l'envoi d'un message MPI. La fenêtre de congestion nous indique si un message va être envoyé en une ou plusieurs fois, ce qui va introduire du délai. Cependant cette information n'est pas suffisante car elle ne nous donne que le temps où les segments TCP sont retenus du coté de l'émetteur. L'autre information utile est le nombre et le moment des retransmissions car elle précise la raison des délai introduits dans l'émission d'un message.

Ces informations peuvent être obtenues conjointement en étudiant le tampon d'émission de TCP. En effet, la quantité de données présentes dans les tampons est caractéristique du temps que vont mettre ces données à traverser la pile TCP. Lorsque un `write` est posté sur une socket, les données sont copiées dans le tampon d'émission de TCP. Ces données seront supprimées au fur et à mesure de la réception des acquittements de la part du récepteur. Ainsi, lorsque le tampon d'émission est vide, on est sûr que le message a été reçu en totalité par le récepteur. Le durée entre le moment où le message arrive dans le tampon et le moment où ce tampon

[1] La norme numéro 60027-2 de la Commission Electronique Internationale (CEI) stipule que les préfixes binaires soient abrégés Ki, Mi et Gi pour les différentier des préfixes décimaux. Ceux-ci correspondent respectivement à 2^{10}, 2^{20} et 2^{30} octets.

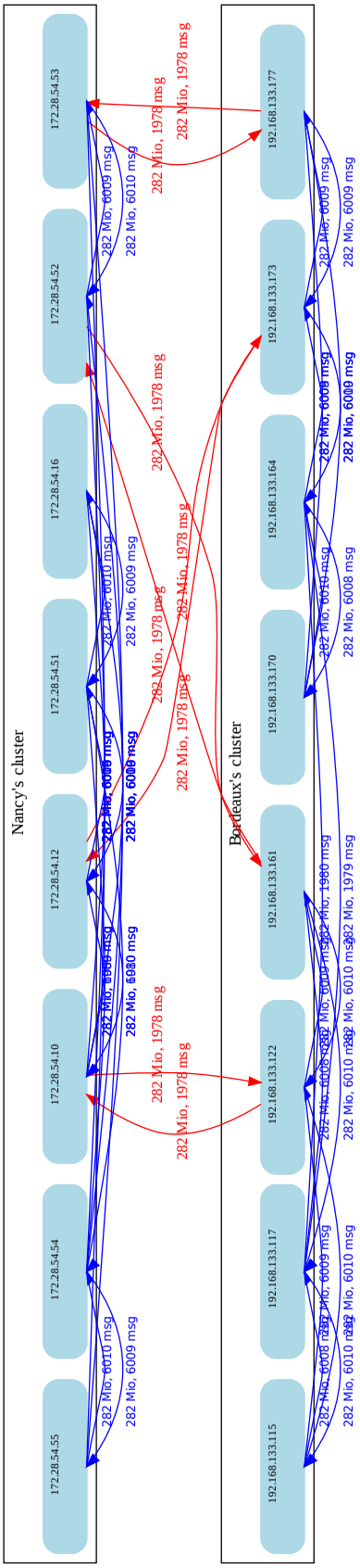
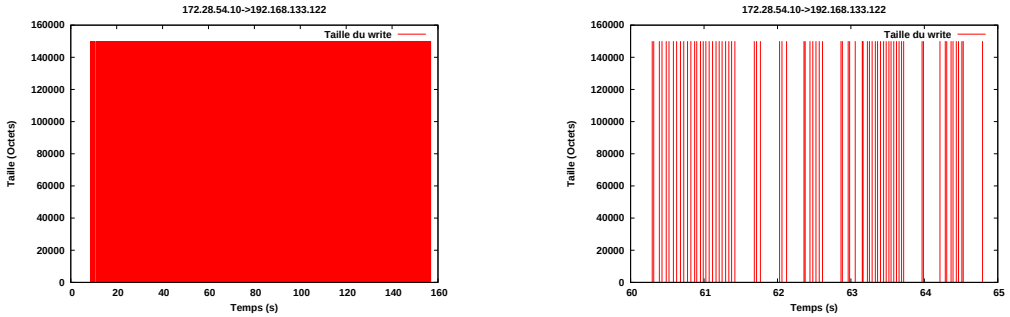


FIG. 3.1 – Schéma de communication de CG



(a) Vue globale (b) Vue détaillée sur 5s

FIG. 3.2 – Détail des *write* au cours du temps lors de l'exécution de CG

est vide, représente donc le temps d'émission du message. Si la fenêtre de congestion est le facteur limitant, les données vont rester plus longtemps dans le tampon. Elles augmenteront donc ce temps d'émission. De la même manière, si une perte a lieu, les données mettront plus de temps avant d'être supprimées de ce tampon.

Le module `tcpprobe` enregistre l'état d'une connexion TCP lors de l'arrivée de paquets (données ou ACK). Il fonctionne par l'insertion d'un détournement dans la pile de réception de TCP de manière à capturer la fenêtre de congestion. Nous avons donc modifié le module `tcpprobe` afin d'obtenir également les informations supplémentaires dont nous avions besoin : le moment des retransmissions et l'évolution de la quantité de données présente dans le tampon d'émission.

3.1.4 Récupération des caractéristiques des Nas Parallel Benchmarks (NPB)

Les outils que nous avons décrit dans les deux sections précédentes permettent d'obtenir des informations sur les communications des applications MPI. Nous avons utilisé InstrAppli pour obtenir les caractéristiques des NAS Parallel Benchmark, un ensemble de programmes représentatifs des applications de grille, que nous utilisons dans nos tests.

3.1.4.1 Description des NPB

Les Nas Parallel Benchmarcks (NPB) sont un ensemble de 8 programmes (BT, CG, EP, FT, IS, LU, MG and SP) qui donnent un aperçu représentatif des applications parallèles qui peuvent s'exécuter sur une grappe ou sur une grille. Les données fournies en entrée des NPB correspondent à des problèmes de différentes tailles classées en six classes (S, W, A, B, C, D). Pour nos mesures, nous avons utilisé la classe B entre 4 ou 16 nœuds.

Chaque programme a un schéma de communication particulier dépendant de la classe et du nombre de nœuds mais pas du réseau d'exécution. L'article [MTM+09] étudie le profil de ces applications la similarité des applications BT, LU, MG et SP, selon la classe et le nombre de noeuds. Ainsi, BT et MG sont similaires selon la taille des classes mais SP et BT varient dans la quantité de données quelles envoient. L'article [FY02] donne le type de communications et le nombre de messages pour la classe A sur 16 nœuds. Nous avons fait le même type d'expériences avec l'outil InstrAppli présenté à la section précédente pour obtenir le schéma des communications ainsi que le nombre de `write`, la taille des messages et le débit utilisé sur le lien longue-distance uniquement.

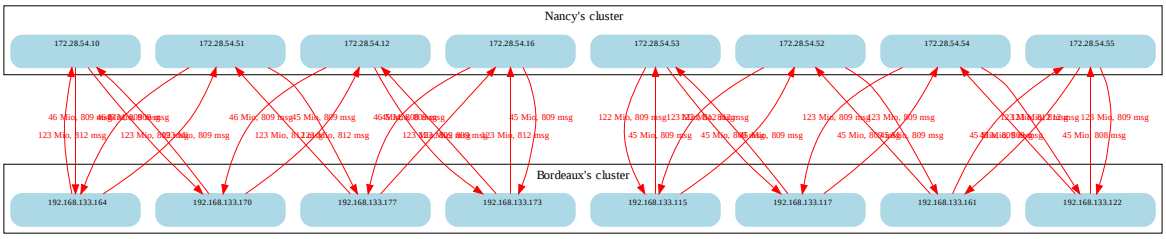


FIG. 3.3 – Schéma de communication de BT

3.1.4.2 Caractéristiques de communications des NPB

Nous avons exécuté les NPB de manière à récupérer à l'aide de InstrAppli leur schéma de communication et les caractéristiques de leurs communitations. Ceux-ci sont nécessaires pour comprendre et analyser les pertes de performance comme il est fait à la fin de ce chapitre et dans les chapitres 4 et 5.

De la même manière que la figure 3.1 présentait le schéma de communication de CG, les figures 3.3 et 3.4 présentent deux autres exemples de schémas de communications des NPB. Les communications locales ont été omises dans un souci de clarté. Pour BT, chaque noeud communique avec deux autres noeuds du site distant alors que dans FT chaque noeud communique avec tous les autres à l'aide de la fonction `MPI_Alltoall`.

La figure 3.5 présente pour chaque NPB un exemple des communications longue distance qui sont effectuées entre deux noeuds. FT et IS par exemple communiquent peu de fois mais avec de gros messages. MG communique avec des messages de tailles périodiquement croissante. Les quatre autres NPB (BT, CG, LU et SP) communiquent constamment, avec une alternance des tailles des messages pour BT et SP et, une période sans communication pour LU.

Le tableau 3.1 présente les caractéristiques des communications longue distance des NPB. Les valeurs sont l'aggrégation de toutes les connexions longue distance. On peut distinguer deux groupes de programmes : BT, CG, EP, LU, MG et SP utilisent des fonctions point à point alors que FT et IS communiquent grâce à des opérations collectives. EP fait principalement du calcul et ne communique que très peu. L'intérêt de ce NAS étant uniquement de tester les capacité de calcul, il ne sera pas employé par la suite. CG et MG communiquent avec des messages de tailles variées. LU envoie des messages de taille moyenne. BT et SP communiquent avec de nombreux messages de grande taille. Dans le second groupe, FT et IS communiquent à l'aide d'opérations collectives : FT utilise la primitive `MPI_Alltoall` et IS les primitives `MPI_Allreduce` et `MPI_Alltoallv`.

3.1.4.3 Classification des NAS

InstrAppli permet d'obtenir les schémas de communication des applications, l'évolution dans le temps des communications et les différents paramètres du tableau 3.1 qui présente la quantité de données transmise sur le WAN, le nombre de connexions, la taille des `write` et leur fréquence. Même s'il est difficile de déterminer le temps de calcul par rapport au temps de communication, les informations obtenues nous permettent d'établir une classification. Celle ci se base sur (1) la fréquence des communications, (2) la taille des `writes` et (3) l'aspect synchrone des communications.

Concernant, la fréquence des communications, la dernière colonne du tableau 3.1 nous permet de déterminer le classement suivant : MG, LU et CG communiquent souvent, SP et

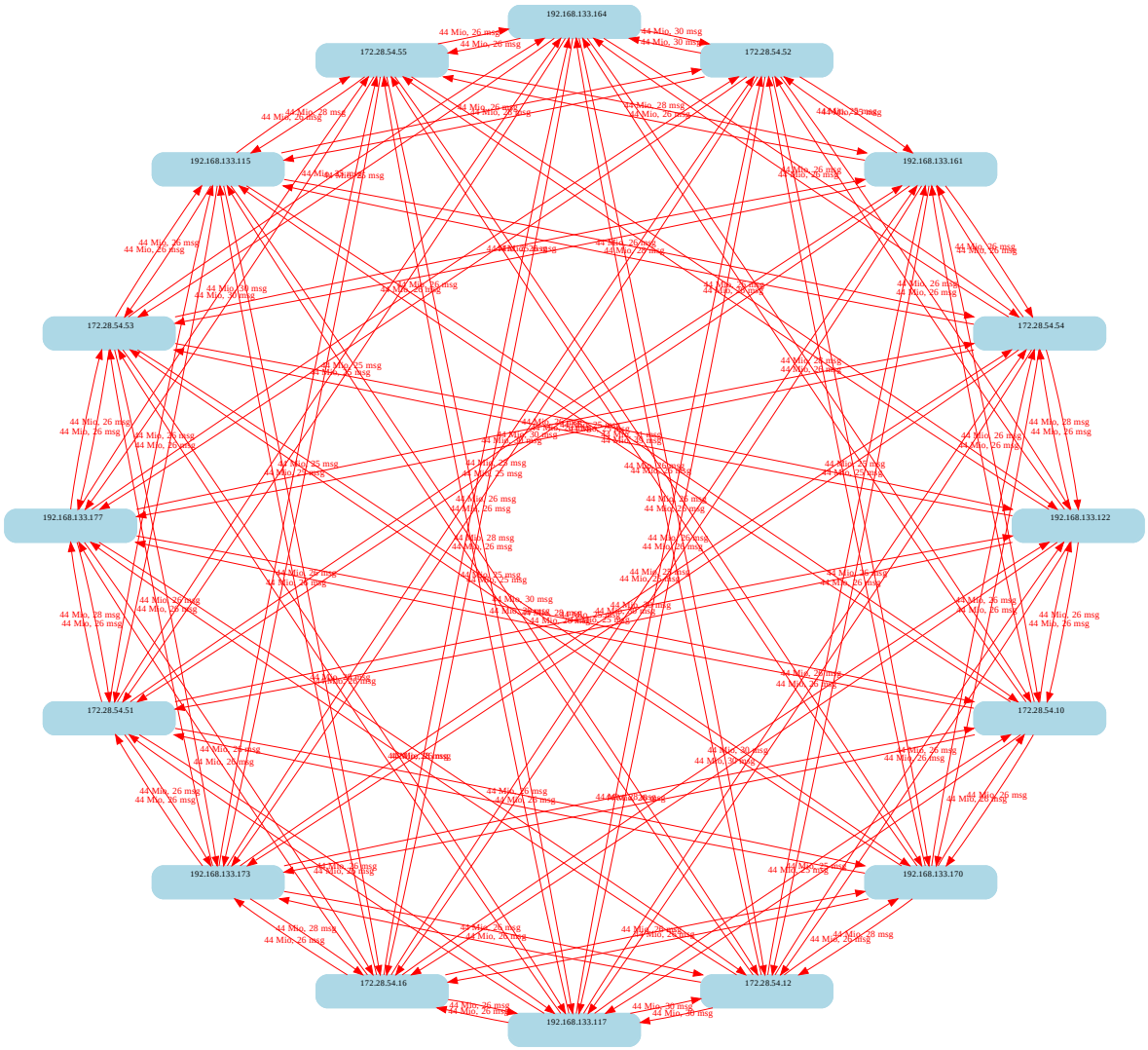
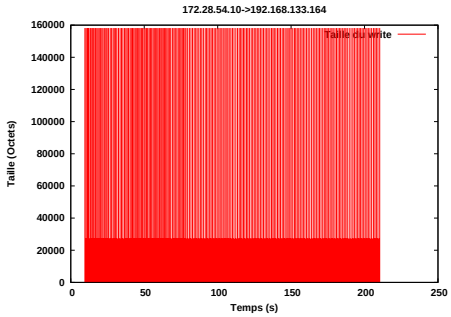
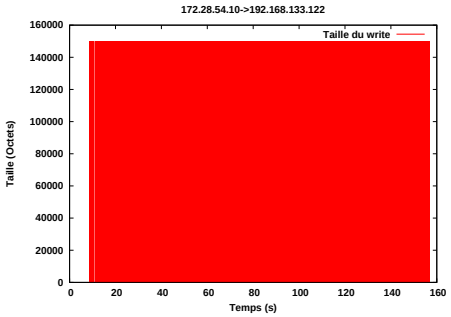


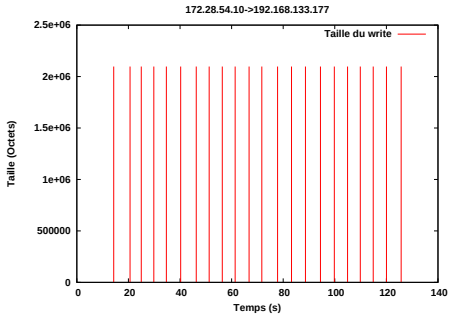
FIG. 3.4 – Schéma de communication de FT



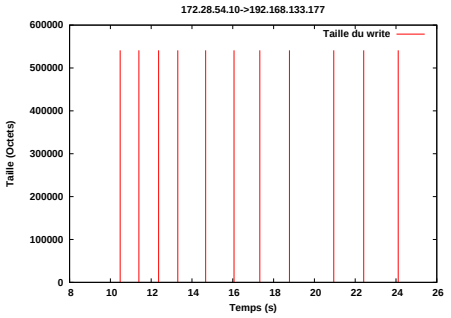
(a) BT



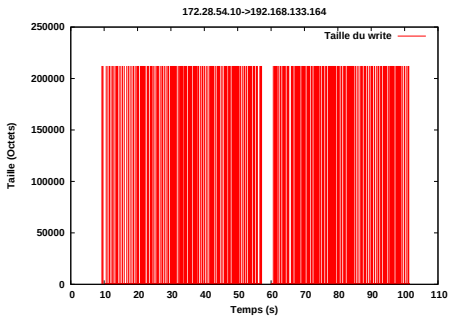
(b) CG



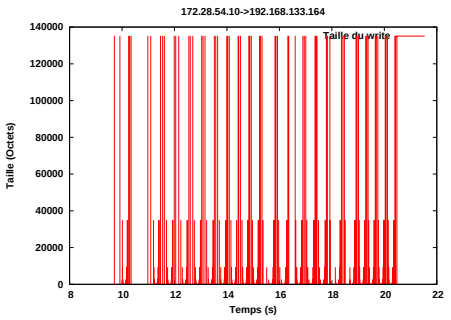
(c) FT



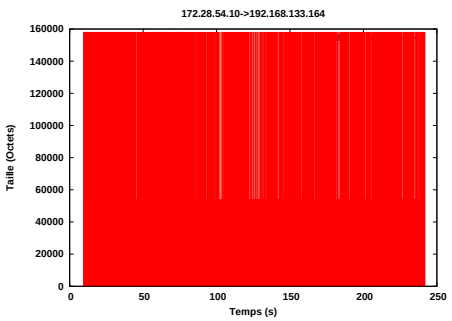
(d) IS



(e) LU



(f) MG



(g) SP

FIG. 3.5 – Evolution temporelle des communications longue distance entre deux noeuds lors de l'exécution des différents NPB

	Type de comm.	Somme des communications sur le WAN		Nombre de connexions WAN	Tps d'exéc sur une grappe	Temps moyen entre deux write (en s)
		taille	quantité			
BT	P. à Point	9648 writes de 26 Kio + 16112 w. de 160 Kio	2.8 Gio	32	104 s	0.248
CG	P. à Point	15800 writes de 150 Kio	2.37 Gio	8	26 s	0.075
FT	Collective	600 writes < 200 o + 2816 writes > 2 Mio	5.9 Gio	240	34 s	4.59
IS	Collective	1151 writes < 400 o + 0.5 Mio < 1400 writes < 0.6 Mio	0.74 Gio	240	3 s	1.24
LU	P. à Point	200000 writes de 1 Kio + 2000 w. de 200 Kio	0.62 Gio	8	62 s	0.004
MG	P. à Point	8842 * variées de 40 o à 135 Kio	0.19 Gio	24	3 s	0.026
SP	P. à Point	45 Kio<19248 writes<54 Kio + 100 Kio<32112 w.<160 Kio	5.1 Gio	32	92 s	0.145

TAB. 3.1 – Caractéristiques des communications des NPB sur le réseau longue distance. Les données sont aggrégées sur toutes les connexions longue distance.

BT assez peu souvent et, IS et FT très peu souvent.

La taille des writes nous permet de déterminer si une application communique beaucoup ou non. Ainsi, LU est une application qui n'écrit que de petits messages, FT et IS émettent de gros messages et, BT, CG, MG et SP sont intermédiaires.

Enfin, la dernière classification concerne l'aspect synchrone des communications même si celle-ci n'est pas donnée par les informations précédentes. Comme elles utilisent des opérations collectives, FT et IS sont typiquement dans cette catégorie. Certaines phases de MG sont synchrones comme il est décrit dans [CDS00]. Enfin, CG se synchronise avec ses voisins pour effectuer ses communications (comme il est décrit dans [Pro]).

Cette classification nous permet d'étudier l'impact des différents point problèmatiques de la grille sur ces applications.

3.2 Evaluation de l'exécution des applications MPI sur la grille

Les outils que nous avons présenté en sections 3.1.3.1 et 3.1.3.2 nous ont permis d'obtenir une bonne vision des communications des applications MPI en général et des NAS Parallel Benchmark en particulier. Il est maintenant nécessaire de comprendre comment ces applications interagissent avec leur environnement de communication. Nous allons dans un premier temps évaluer l'ensemble des liens de notre plateforme d'exécution, Grid5000. La seconde partie évalue les performances des NPB sur Grid5000 en détaillant l'impact des trois points problématiques évoqués précédement : la latence, le goulot d'étranglement du WAN et le partage du lien longue distance par les différentes applications de la grille.

3.2.1 La grille Grid'5000

Nos expériences ont été menées à l'aide de la grille de recherche Grid'5000. La figure 3.6 représente l'interconnexion des 9 sites de la grille (Bordeaux, Grenoble, Lille, Lyon, Nancy, Orsay, Rennes, Sophia et Toulouse) répartis sur toute la France. Les sites sont interconnectés par un réseau longue distance avec des liens à 1 ou 10 Gbit/s opérés par RENATER [Sita]. Les latences entre les sites sont variées. Par exemple, la latence TCP entre Lyon et Grenoble est de 4 ms alors que celle entre Rennes et Sophia est de 19 ms. Le nombre de coeurs et de processeurs des nœuds est également variable conduisant à des puissances de calcul différentes.

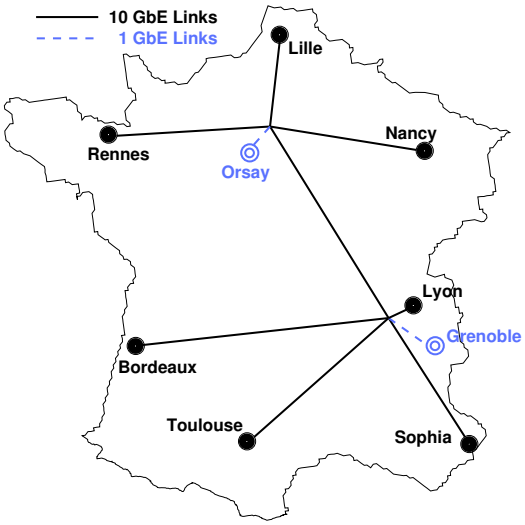


FIG. 3.6 – Réseau d'interconnexion de Grid'5000

Les réseaux rapides locaux sont hétérogènes (Infiniband, Myrinet, Ethernet 1 ou 10 Gbit/s) et certaines machines possèdent plusieurs de ces interfaces.

Les utilisateurs doivent réserver les noeuds avant de pouvoir les utiliser. Grid5000 fournit aux chercheurs la possibilité de configurer aisément l'environnement sur lequel s'exécutent leurs expériences grâce à un système de déploiement d'image disque. Celle-ci contient leur système d'exploitation, leurs logiciels et leur configuration qu'ils copieront sur tous les nœuds au début de leur expérience. De cette manière, ils peuvent exécuter leurs applications avec un environnement identique sur tous les nœuds et obtenir les droits d'administration afin de les configurer à leur guise.

Dans le rapport [GHVBPS06], nous avons présenté une évaluation des liens de Grid'5000 à l'aide de deux métriques : la latence entre les sites et le débit maximal atteignable entre ceux-ci. La latence est présentée dans le tableau 3.2. On peut constater que la distance est le principal facteur qui influe sur la latence. Une première évalution des débits obtenus entre les sites montrait des performances hétéroclites (cf. première valeur du tableau 3.3). Nos investigations nous ont conduit à modifier les valeurs des tampons d'émission et de réception de TCP. En effet, TCP stocke les paquets pour pouvoir les retransmettre s'ils sont perdus ou corrompus, puis les détruit à la réception d'un acquittement (ACK). Comme il est décrit dans [SMM98], pour utiliser pleinement les capacités des liens avec TCP, les tampons doivent pouvoir contenir tous les paquets qui peuvent être envoyés pendant un aller-retour entre les deux points de la connexion (c.-à-d. le temps d'un RTT). La taille des tampons doit donc être égale à $debit * RTT$. Les résultats avec les valeurs ajustées sont présentées dans le tableau 3.3 (seconde valeur). La plupart des liens permettent d'obtenir des performances proches de la bande passante maximale (qui est de 941 Mbit/s avec TCP sur des liens 1 Gbit/s du fait des entêtes TCP, IP et Ethernet).

Le taille des tampons de TCP influe donc directement sur les performances. Si leur taille est trop petite, ils conduisent à une sous utilisation des liens. Suite à nos travaux, ces valeurs ont été modifiées sur les environnements par défaut de Grid'5000. Ce sera aussi le cas pour toutes les expériences dont nous présentons les résultats par la suite.

		Source								
		Bordeaux	**Grenoble**	**Lille**	**Lyon**	**Nancy**	**Orsay**	**Rennes**	**Sophia**	**Toulouse**
Destination	**Bordeaux**		16.1	11.59	9.96	12.73	8.86	8.06	10.6	3.95
	Grenoble	16.04		12.83	3.3	13.24	15.15	15.22	9.87	11.36
	Lille	11.61	12.84		10.16	9.19	4.54	11.23	16.78	18.51
	Lyon	9.99	3.28	10.17		10.57	9.27	12.57	7.22	8.70
	Nancy	12.72	13.26	9.19	10.57		5.72	11.63	17.19	18.62
	Orsay	8.85	15.16	4.53	9.26	5.72		9.03	20.38	12.4
	Rennes	8.04	15.22	11.23	12.56	11.63	9.03		19.18	20.65
	Sophia	10.60	9.96	16.78	7.22	17.19	20.38	19.17		15.25
	Toulouse	3.80	11.61	18.24	8.64	18.66	12.4	20.66	15.86	

TAB. 3.2 – Latences observées sur Grid5000 (en secondes).

		Source								
		Bordeaux	**Grenoble**	**Lille**	**Lyon**	**Nancy**	**Orsay**	**Rennes**	**Sophia**	**Toulouse**
Destination	**Bordeaux**		58.1/771	61.8/725	55.9/862	81.2/911	111/884	76.3/852	68.9/875	181/685
	Grenoble	32.3/900		34.0/701	151/925	39.8/812	33.7/893	34.3/787	52.6/911	48.4/647
	Lille	53.3/738	70.0/838		53.6/120	112/922	199/848	55.0/916	44.3/598	33.9/579
	Lyon	61.5/425	230/912	71.2/786		97.6/904	106/740	49.8/864	100/926	72.0/730
	Nancy	48.0/725	162/851	78.5/742	52.4/865		777/854	54.7/938	43.3/931	32.0/622
	Orsay	67.8/799	54.1/866	150/777	58.8/869	936/936		68.7849	36.2/878	50.8/523
	Rennes	64.2/912	33.6/831	46.6/787	41.4/859	45.5/914	56.5/912		27.4/839	26.3/651
	Sophia	47.0/901	46.1/839	29.5/653	67.4/543	28.9/611	22.3/900	25.1/321		34.0/694
	Toulouse	166/928	47.6/859	29.8/784	65.7/882	29.7/933	44.3/923	26.3/939	36.1/909	

TAB. 3.3 – Bande passante entre les sites de Grid'5000 avant/après optimisation de la taille du tampon TCP (en Mb/s).

3.2.2 Banc d'essai utilisé dans nos expériences

La plupart de nos expériences ont été menées sur Grid'5000, à l'aide d'un même banc d'essai, composé de deux sites séparés par le réseau longue distance. La figure 3.7 représente un schéma générique de cet environnement paramètré par n, RTT et bdp, qui seront précisés à chaque expérience. Nous utilisons le même nombre de noeuds n sur les deux sites s_1 et s_2. $N_{i.j}$ est le noeud j du site i. La puissance des noeuds est différente d'un site à l'autre et peut grandement faire varier les performances selon que l'on se trouve sur des machines avec 2 ou 8 cœurs par exemple mais les comparaisons sont effectuées de manière unitaire c.-à-d. avec les même machines. bdp représente la capacité du lien longue distance à 1 ou 10 Gbit/s. Le RTT varie selon les sites choisis (de 4 à 18 ms) mais sera précisé dans chacun des cas. Les noeuds sont reliés par des cartes Ethernet à 1 Gbit/s. Dans certains cas, nous utiliserons des noeuds avec deux cartes Ethernet reliées au même commutateur, celles-ci étant représentées par G_s sur la figure (G pour "*gateway*"). Nous exécutons toujours un seul processus par noeud. Ce modèle de type "*dumpbell*" est le modèle classique d'étude des problématiques TCP.

3.2.3 Utilisation de la grille comme environnement d'exécution : impact des points problématiques

A l'aide de l'environnement que nous venons de décrire, nous étudions la question Q1 : la grille est-elle un environnement valable pour l'exécution d'applications MPI ? Nous étudions tout d'abord dans quelle mesure le fait d'augmenter la capacité de calcul en utilisant la grille permet d'améliorer les temps d'exécution des NPB par rapport à ceux obtenus sur une seule

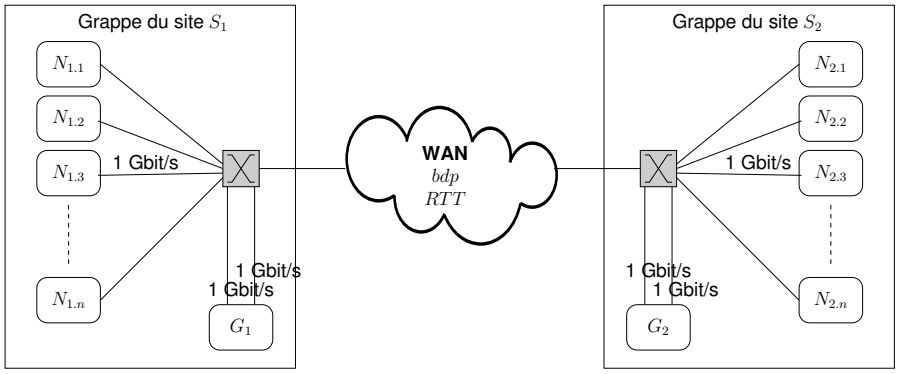


FIG. 3.7 – Banc d'essai générique utilisé dans nos expériences

grappe. Ensuite, nous évaluons comment les trois points problématiques introduits par le réseau longue distance, à savoir la latence plus élevée, le goulot d'étranglement et le partage de la bande passante disponible, pouvaient impacter les applications.

3.2.3.1 Comparaison de l'exécution sur une grappe par rapport à la grille

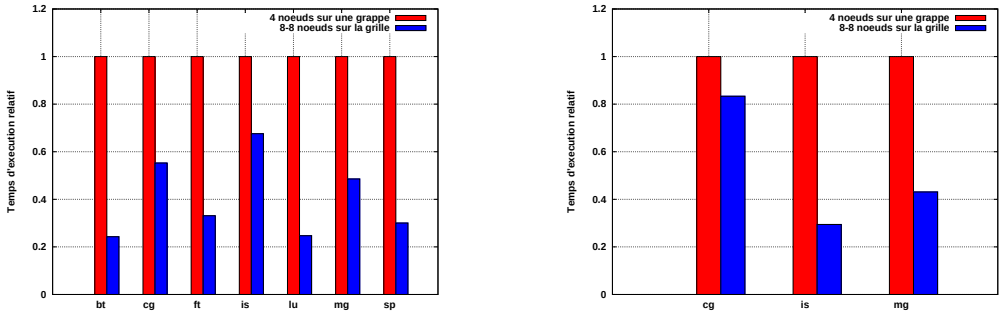
La grille propose d'aggréger plusieurs grappes distribuées afin de disposer d'une plus grande puissance de calcul. Mais les contraintes imposées par cet environnement sont fortes. Il est nécessaire de se demander si l'augmentation du nombre de ressources grâce à l'utilisation de la grille permet d'améliorer les temps d'exécution des applications qui s'exécuteraient sur une seule grappe avec moins de ressources mais des communications plus performantes.

L'expérience qui suit a été exécutée sur le banc d'essai présenté en section 3.2.2 avec des noeuds de même puissance reliés par un réseau à $10\,Gbit/s$ et une latence de $11.6ms$.

La figure 3.8(a) présente les temps d'exécution sur 8-8 noeuds de la grille relativement aux temps sur une grappe de 4 noeuds. La taille des problèmes est celle de la classe B des NPB et elle est identique quelque soit le nombre de processus utilisés. On constate tout d'abord que toutes les applications améliorent leurs performances sur la grille. Cependant, si le gain était proportionel au nombre de noeuds (speedup linéaire), les performances devraient idéalement s'approcher de 0.25. BT, LU, SP et FT obtiennent des performances proches de ce maximum. A contrario, pour les autres applications (CG, IS et MG), le bénéfice de la grille est moindre. Ces applications communiquent trop par rapport à leur temps d'exécution qui est très court.

Pour valider cette hypothèse, nous avons augmenté la taille du problème pour ces applications. La figure 3.8(b) présente les résultats d'une expérience similaire à la précédente mais avec la classe C des NPB. Cette fois ci, on s'aperçoit que le temps d'exécution de IS et MG, est fortement diminué sur la grille. CG par contre ne parvient toujours pas à en exploiter les ressources. Ceci est dû à son mode de communication qui synchronise une partie des processus entre eux comme il est décrit dans [Pro]. Les processus ne sont pas différenciés selon s'ils sont sur un site ou l'autre de la grille. De ce fait, le placement des processus sur les différents sites n'étant pas optimal, la synchronisation des communications longue distance contribue à ralentir la totalité de l'application par alignement sur la plus longue latence.

L'augmentation des ressources ne bénéficie pas également à toutes les applications. Du fait du coût des communications sur la grille, le gain observé n'est pas proportionnel à l'augmentation du nombre de noeuds. Au delà de la latence entre les sites qui est incompressible et pénalisante, nous pensons qu'il est possible d'améliorer l'utilisation de la grille par les applications MPI. Nous avons donc cherché à comprendre quel était l'impact de l'implémentation,



(a) Class B (b) Class C

FIG. 3.8 – Comparaison des temps d'exécution des NPB sur 8-8 noeuds de la grille normalisé à 4 noeuds sur une grappe.

l'impact de la latence, du goulot d'étranglement du WAN et du partage du réseau sur les performances de ces applications. Nous étudions donc dans la suite la variation de chacun de ces quatre paramètres séparément.

3.2.3.2 Comparaison des implémentations MPI dans un réseau de grille

Comme vu dans l'état de l'art du chapitre 2, les différentes implémentations disponibles pour la grille ne proposent pas toutes les mêmes optimisations. Nous avons donc cherché à comparer les plus à jour d'entre elles : MPICH2, GridMPI, MPICH-Madeleine, OpenMPI et MPICH-G2 ont été évaluées à l'aide d'un pingpong et des NPB.

Nous avons donc exécuté ces applications sur deux sites différents, Nancy et Rennes, séparés par un WAN à $10\,Gbit/s$ avec une latence de $11.6\,ms$.

Nos premières expériences ont porté sur un simple pingpong pour comparer les performances des communications point-à-point au niveau MPI à celle obtenues sur TCP. Le tableau 3.4 présente les latences obtenues au niveau MPI en utilisant chacune des implémentations (temps pour un aller entre deux noeuds). La différence par rapport à TCP est mise entre parenthèse. Toutes les implémentations présentent un surcoût similaire, de l'ordre de $5\,\mu s$ que ce soit à l'intérieur d'un cluster ou sur la grille, excepté MPICH-Madeleine et MPICH-G2. Pour MPICH-G2, ce surcoût est sans doute dû à la gestion des certificats et à l'authentification des connexions. La latence totale est bien évidemment supérieure sur la grille due à la distance physique. De ce fait, le surcoût dû aux implémentations devient négligeable.

La bande passante obtenue est visible sur la figure 3.9. Les deux implémentations destinées aux grilles, GridMPI et MPICH-G2, parviennent à obtenir des performances similaires à celle de TCP. MPICH-G2 souffre pourtant pour l'envoi de messages supérieurs à 4M. Les trois autres implémentations présentent un seuil dû au changement de mode d'envoi : passage du mode `eager` au mode `rendez-vous`.

Pour comparer ces implémentations à l'aide d'applications plus conséquentes, la suite de nos expériences a porté sur les NPB exécutés sur 8 noeuds de chacun des clusters. Chaque NPB est exécuté six fois, la valeur retenue étant le minimum des temps d'exécution pour s'abstraire des perturbations dues aux autres utilisateurs de la grille.

La figure 3.10 montre le temps d'exécution des NPB pour chaque implémentation normalisé à MPICH2. Une valeur inférieure représente une diminution du temps d'exécution. Les perfor-

	Dans la grappe de Rennes	Dans la grille : entre Rennes et Nancy
TCP	41	5812
MPICH2	46 (+5)	5819 (+7)
GridMPI	46 (+5)	5820 (+8)
MPICH-Mad.	62 (+21)	5826 (+14)
OpenMPI	46 (+5)	5821 (+9)
MPICH-G2	79 (+38)	5850 (+38)

TAB. 3.4 – Comparaison de la latence au niveau application avec différentes implémentations dans un cluster et dans une grille (en μs).

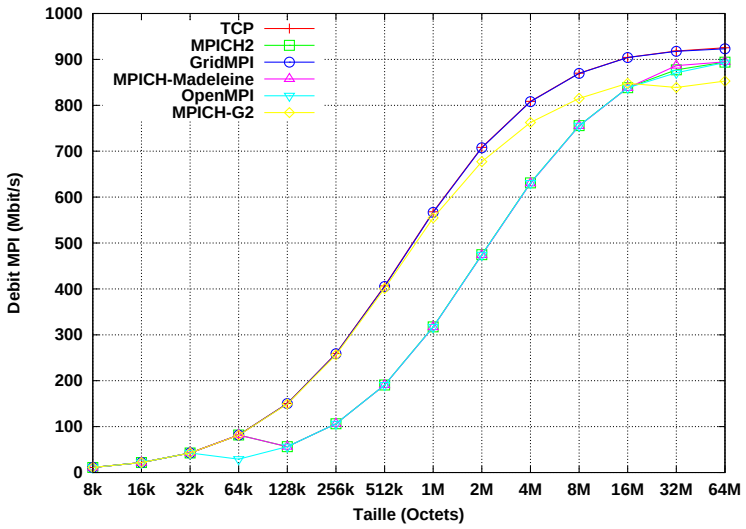


FIG. 3.9 – Comparaison de la bande passante au niveau application avec différentes implémentations dans une grille (en $Mbits/s$).

mances sont assez disparates. Le temps obtenus avec MPICH2 sont légèrement meilleurs pour LU. Pour MG, GridMPI, MPICH-Madeleine et OpenMPI présentent de bonnes performances. Pour CG, GridMPI et MPICH-G2 divisent le temps d'exécution par deux. Enfin, GridMPI montre des performances légèrement meilleures avec BT et SP. Les résultats pour BT et IS n'ont pas pu être obtenus avec MPICH-Madeleine (le programme ne termine pas).

Pour FT et IS, la diminution du temps d'exécution avec GridMPI est importante sur la grille dû à ses opérations collectives. En effet, les autres implémentations proposent des optimisations efficaces pour les grappes. Or, celles-ci ne sont plus utiles et même pénalisantes dans le cas des grilles. Afin de nous rendre compte de cette limitation, nous avons executé les IMB (Intel MPI Benchmark, [IMB]) qui proposent de regarder les temps d'exécution de chacune des fonctions de communications de MPI. La figure 3.2.3.2 représente le temps d'exécution d'un Alltoall en local (figure 3.11(a)) et entre deux sites (figure 3.11(b)), dans la même configuration que précédemment. Alors que les résultats en local sont similaires pour toutes les implémentations, les résultats sur la grille montrent clairement un changement de comporte-

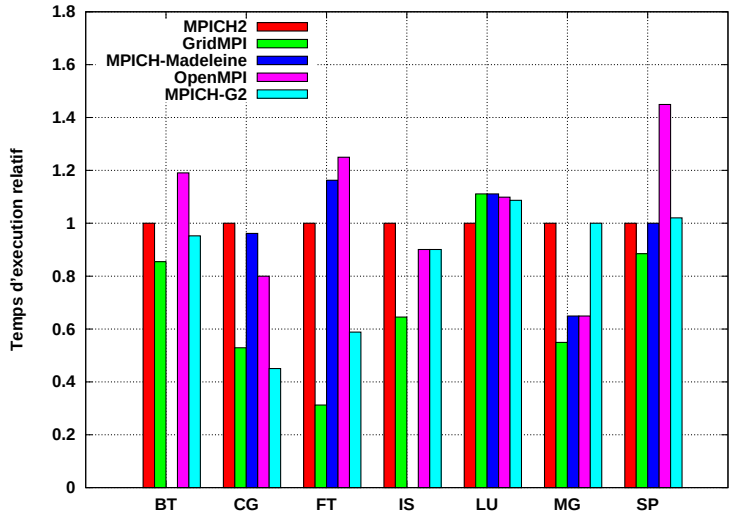
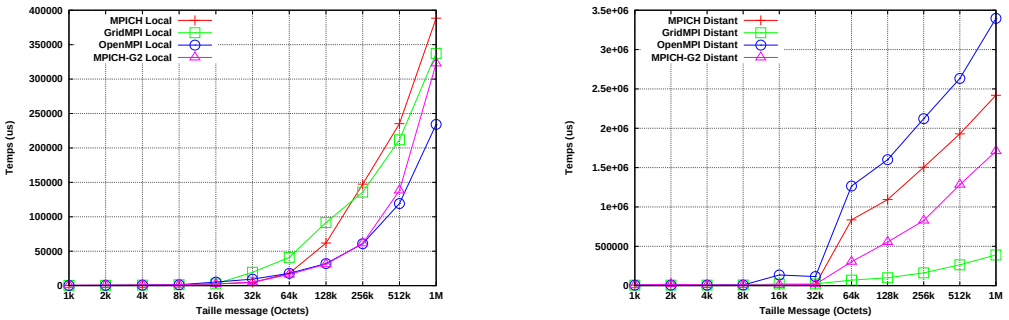


FIG. 3.10 – Comparaison relative des différentes implémentations sur 8-8 nœuds sur deux grappes, MPICH2 est la référence.

ment à 32 Kio pour toutes les implémentations excepté GridMPI. En effet, les implémentations changent d'algorithme d'envoi des messages selon une taille fixée dans le code. Cependant, si ce seuil est optimal pour une utilisation au sein d'un cluster, il n'est pas approprié pour un réseau longue distance à forte latence. GridMPI prend ce paramètre en compte et de ce fait, les performances de GridMPI sont très bonnes sur la grille pour les programmes qui utilisent des opérations collectives tels que FT et IS. De même, l'ordre du temps d'exécution des *Alltoall* est identique à celui de l'exécution de FT : GridMPI, MPICH-G2, MPICH2, puis OpenMPI (par ordre croissant des temps d'exécution).

Même s'il est difficile de généraliser, GridMPI semble l'implémentation la plus efficace sur les grilles particulièrement pour les applications qui utilisent des opérations collectives. Elle est suivie par MPICH-G2. Cependant, afin de conserver une cohérence entre toutes les parties,



(a) Local (b) Distant

FIG. 3.11 – Résultats du Alltoall des IMB

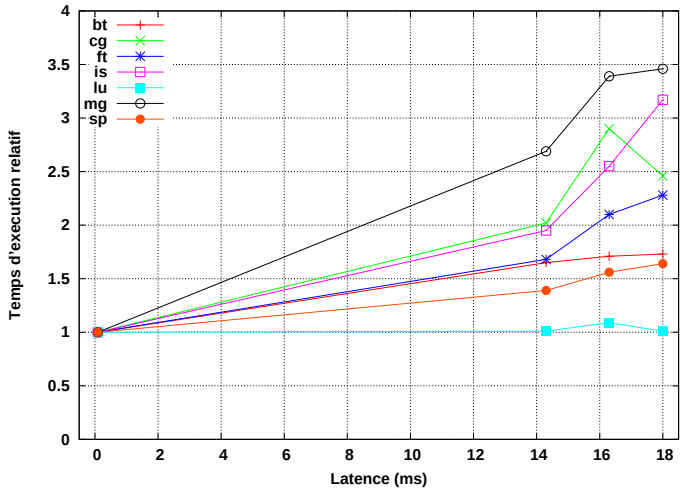


FIG. 3.12 – Evolution du temps d'exécution des NPB en fonction de la latence normalisés au temps sur une grappe.

nous avons conservé MPICH2 pour toutes les autres expériences.

3.2.3.3 Impact de la latence

Pour cette expérience, nous avons utilisé deux configurations différentes : une grappe de 16 nœuds (en utilisant $N_{1.1}$ à $N_{1.16}$, cf Fig. 3.7) ou deux grappes de huit nœuds reliés par un WAN (en utilisant $N_{1.1}$ à $N_{1.8}$ et $N_{2.1}$ à $N_{2.8}$). Les machines physiques sont identiques et la latence varie selon les sites utilisés (14.3, 16.3 et 18 ms).

La figure 3.12 montre le surcoût du temps d'exécution des NPB en fonction de la latence relatif au temps sur une grappe. Une valeur plus élevée correspond à un temps d'exécution supérieur.

Tous les NPB présentent un surcoût dû à l'introduction de latence dans les communications. Cependant, toutes les applications ne sont pas impactées de la même manière. LU n'est que très peu impactée (surcoût inférieur à 10%) parce qu'elle ne communique pas beaucoup. SP et BT (surcoût inférieur à 60%) sont également peu impactées parce qu'elles communiquent peu souvent. MG communique souvent, elle est donc très impactée par l'augmentation de la latence (surcoût de plus de 250%). IS qui communique beaucoup avec un temps d'exécution relativement faible est très fortement impactée par l'augmentation de la latence. Enfin, CG et FT présentent un surcoût intermédiaire. Nous n'expliquons pas l'augmentation du surcoût pour CG à 16 ms de latence.

La latence influe donc à la fois sur les applications qui communiquent souvent et sur les applications qui communiquent beaucoup dans un temps court.

3.2.3.4 Impact du goulot d'étranglement du WAN

L'expérience suivante avait pour but de comparer l'impact du goulot d'étranglement que constitue le WAN. La configuration utilisée était de 8 noeuds sur chaque site. Afin de faire varier le goulot d'étranglement, les noeuds sont connectés avec deux cartes réseau, chacune sur

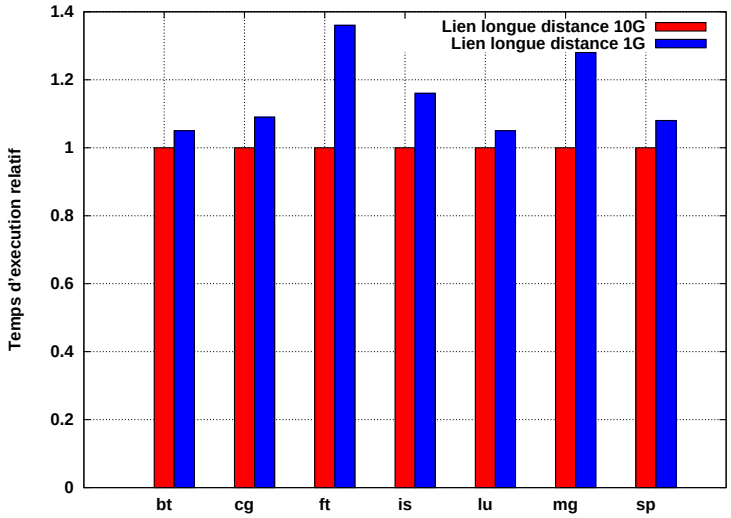


FIG. 3.13 – Comparaison relative des temps d'exécution avec un lien longue distance à 1 $Gbit/s$ et 10 $Gbit/s$. Les temps à 10G $Gbit/s$ sont la référence.

un commutateur différents : l'un est connecté au réseau longue distance par un lien à 1 $Gbit/s$, l'autre par un lien à 10 $Gbit/s$. La latence entre les sites est de 16.3 ms.

La figure 3.13 présente les temps d'exécution avec un lien longue distance à 1 $Gbit/s$ relatifs au temps d'exécution avec un lien à 10 $Gbit/s$. Une valeur plus élevée représente une augmentation du temps d'exécution lorsque que le goulot d'étranglement diminue.

Les applications étudiées sont assez peu impactées par la réduction de bande passante sur la longue distance ; le surcoût est dans tous les cas inférieur à 40%. Les applications qui souffrent le plus sont FT, IS et MG. Cela s'explique par l'aspect synchrone des communications de ces applications comme nous l'avons vu précédemment.

3.2.3.5 Impact du trafic concurrent

Cette expérience a pour but d'étudier l'impact du partage de la grille par différentes applications. Dans notre cas, on aurait pu utiliser d'autres applications MPI pour générer le trafic concurrent mais cela aurait posé des problèmes de compréhension des résultats obtenus car on ne maitrise pas la superposition des trafics des deux applications. Aussi, pour s'abstraire de ces contraintes, la trafic concurrent est réalisé par des flux `iperf` [TGQ+]. `Iperf` a pour but d'évaluer la bande passante disponible sur un lien en envoyant en continu du trafic (UDP ou TCP) sur le lien de manière à déterminer le débit. Il peut également être utilisé pour générer un trafic concurrent maitrisable.

Nous utilisons toujours le même type de banc d'essai avec une bande passante à 10 $Gbit/s$ et une latence à 11.3 ms. Nous utilisons dix-huit machines par site : huit sur lesquelles sont exécutés les processus MPI et dix machines supplémentaires sur lesquelles nous exécutons l'application `iperf` qui génére un flux TCP à 1$Gbit/s$ sur chaque noeud. Le taux de congestion est défini par : $C_g(n) = \frac{n*D}{bdw}$, où n est le nombre de machines, D leur débit théorique, bdw la capacité nominale du lien longue distance. Le nombre de machines utilisées dont le débit

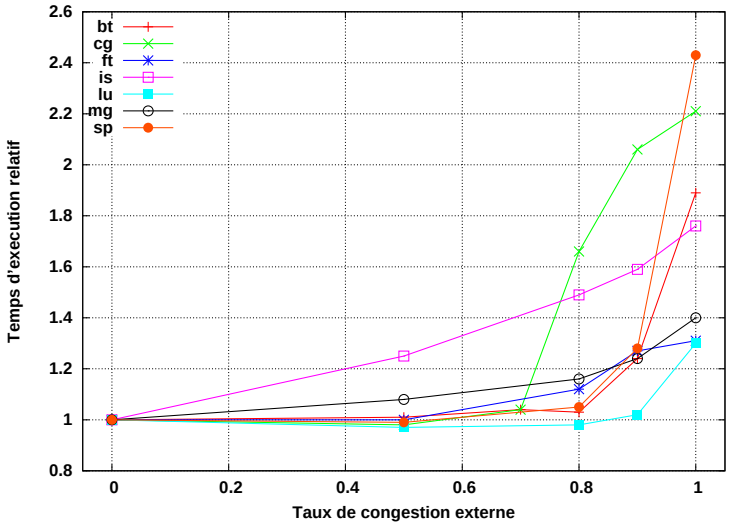


FIG. 3.14 – Comparaison relative des temps d'exécution des NPB avec des taux de congestion croissants. Les temps sans trafic concurrent sont la référence.

nominal est $1\,Gbit/s$ varie de 0 à 10 ; le taux de congestion est donc compris entre 0 et 1 pour un lien longue distance à $10\,Gbit/s$.

La figure 3.14 présente les temps d'exécution relatifs des NPB en présence du trafic concurrent. La référence est le temps d'exécution sans trafic concurrent. Une valeur plus grande indique une augmentation du temps d'exécution quand la congestion augmente.

De manière générale, on peut dire que les applications qui transmettent le plus de données sont les plus impactées par la congestion : BT, CG et SP présentent des surcoût de 90%, 127% et 140%. Cependant, FT fait exception en subissant un surcoût de 30% malgré le fait qu'elle envoie presque 6 Gio de données. Ceci s'explique par le fait qu'elle utilise peu de phases de communication et qu'elle envoie de gros messages. De ce fait, comme nous le montrerons dans la section 4.2, les mécanismes de TCP ont le temps d'adapter le débit d'émission des flux concurrents de manière à laisser la place aux communications MPI. Enfin, nous constatons également que les applications qui communiquent de manière synchrone (FT, IS et MG) ont un surcoût qui augmente linéairement alors que les applications qui communiquent avec des communications point à point (BT, CG, LU, SP) sont essentiellement impactées quand le trafic concurrent utilise tout le lien ($C = 1$). Lorsque les communications d'une application sont synchrones, le facteur limitant est le nombre de flux concurrents sur le lien longue distance. Comme les flux s'aggrègent sur ce lien, à partir du moment où la somme des débits des flux est égale à la bande passante disponible, un flux supplémentaire entraine de la congestion. Lorsque les applications ne sont pas synchrones, l'impact est significatif lorsque le lien ne peut plus absorber le surplus de trafic dû aux applications MPI.

Conclusion

Ce chapitre avait pour but de comprendre les problèmes de performances des applications MPI sur un réseau longue distance. Cette analyse ne pouvait être effectuée sans connaitre les

communications qui étaient effectuées par les applications.

Dans un premier temps, nous avons cherché, dans chacune des couches traversées, les informations nécessaires pour comprendre comment l'application communique. Nous avons présenté deux outils, une librairie et un module noyau, permettant d'obtenir conjointement les paramètres qui pouvaient retarder l'envoi de messages MPI, respectivement au niveau de l'API socket et de la couche TCP. Ces outils nous ont permis d'obtenir les caractéristiques des NPB, le schéma de leurs communications et leur évolution temporelle.

Dans un second temps, à l'aide des informations receuillies dans la première partie, nous avons analysé les performances des NPB sur une grille de calcul. Il ressort que toutes les applications peuvent tirer parti des ressources de la grille. Cependant, le gain est moindre pour celles dont le temps de communication est élevé par rapport au temps de calcul. Comme l'augmentation du nombre de ressources en utilisant la grille ne permet pas d'obtenir un gain linéaire, nous avons étudié l'impact de chaque point problématique de la grille sur les temps d'exécution des applications MPI. La latence diminue principalement les performances des applications qui communiquent souvent. Le goulot d'étranglement et le trafic concurrent affectent principalement les applications avec des communications synchrones (comme les opérations collectives).

La grille est donc un environnement valable pour l'exécution d'applications MPI et permet de trouver des ressources lorsque celles disponibles sur une grappe sont insuffisantes. Cependant, si on compare les temps d'exécution sur une grappe au temps sur la grille, à nombre de noeuds égal il y encore de grande différences (par exemple $104\,s$ contre $222\,s$ pour SP, ou $34\,s$ contre $96\,s$ pour FT). De plus, les informations de ce chapitre, nous montrent que les baisses de performances ne sont pas uniquement liées aux limitations physiques du réseau longue distance (bande passante ou latence entre les sites). Ainsi, le partage du réseau longue distance par l'ensemble des applications de la grille diminue substanciellement le temps d'exécution de certaines applications. Cette interaction est gérée par les mécanismes du protocole de transport. Ceci nous pousse à investiger plus en détails, dans le chapitre suivant, ce qui se passe dans la couche TCP lors de l'envoi de messages MPI sur le réseau longue distance. Nous nous attachons particulièrement à comprendre comment les flux des différentes communications des processus MPI et des autres applications de la grille interagissent.

Chapitre 4

Analyse détaillée de l'interaction entre TCP et les applications MPI

Introduction

TCP a été développé dans le contexte d'Internet pour assurer la transmission fiable des informations dans un réseau où transitent des milliers de flux. C'est actuellement le protocole de transport utilisé pour les communications longue distance dans les réseaux de grille.

Les mécanismes de TCP tels que le contrôle de congestion et le contrôle d'erreurs permettent de limiter la quantité de données émises de manière à garantir une équité statistique entre les flux. Ce sont ces mécanismes qui contrôlent l'interaction entre les différents flux qui circulent sur la grille et assurent le partage de bande passante. Or, comme nous l'avons vu dans le chapitre précédent, les applications MPI sont fortement impactées par les autres flux avec lesquels elles partagent le réseau. Dans ces applications, la quantité de données transmise est beaucoup plus réduite que sur Internet et se présente sous forme de paquets en rafales (envoi de données périodiquement) et non de flux continu (envoi de données en permanence sur le lien, à un débit donné). Dès lors, il est intéressant d'étudier comment les mécanismes de TCP interagissent avec les communications MPI (cf. Q2 section 1.4).

Le temps d'exécution est habituellement la métrique la plus importante pour les applications MPI. Comme celui-ci dépend des performances des communications, il est nécessaire de réduire les temps de transmission des données. Aussi, nous avons étudié au niveau de chacun des mécanismes de TCP, les paramètres qui pouvaient ralentir les communications MPI. Dans un premier temps, nous expliquons comment un délai dans les communications MPI peut ralentir la totalité d'une application. Les deuxième et troisième sections, étudient, respectivement, l'impact du contrôle de congestion et l'impact de contrôle de fiabilité de TCP sur les applications MPI.

4.1 Communications par rafales des applications MPI

Les applications qui utilisent le standard MPI communiquent par passage de messages entre les processus de l'application. Cela signifie qu'elles alternent les phases de calcul et les phases de communications. Elles ne communiquent donc pas continuellement mais envoient des données périodiquement par rafales. De plus, la plupart des applications MPI sont sensibles au délai. On peut voir sur la figure 4.1 l'alternance des phases de calcul et de communication d'une application qui s'exécute sur deux processus. Le processus 1 envoie un message à l'aide de la commande `MPI_Send`. Le processus 2 attend ce message en utilisant un `MPI_Recv` avant de continuer son exécution. Le délai entre le moment où le processus commence à attendre un message et sa réception va donc influer particulièrement sur les performances des applications. Bien que le standard définisse des fonctions non-bloquantes qui pourraient permettre d'éviter l'attente, il n'est pas toujours possible de les utiliser si le processus a justement besoin de ces données pour continuer son exécution.

4.2 Impact du contrôle de congestion de TCP

De par son fonctionnement, le contrôle de congestion limite la quantité de données envoyées sur une connexion TCP pour réduire les pertes. De ce fait, un même message MPI peut être envoyé en plusieurs fois, ce qui en retarde la réception. Or, comme les applications MPI attendent sur les données, tout retard dans leur transmission entraine une augmentation du temps d'exécution.

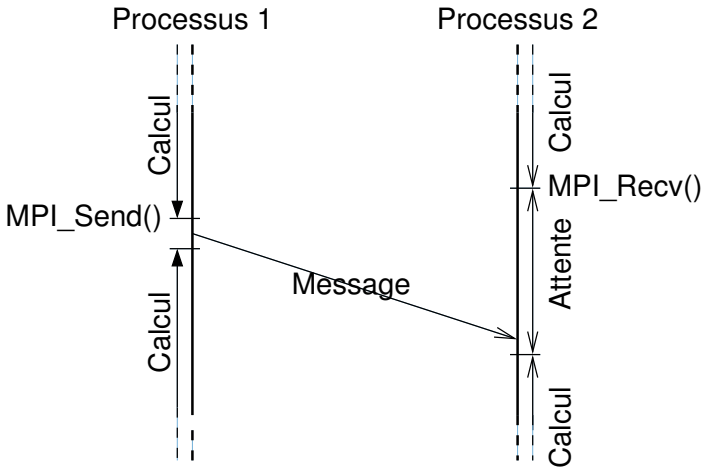


FIG. 4.1 – Phases d'une application MPI.

Cette section s'intéresse aux deux phases du contrôle de congestion : la phase de démarrage lent et la phase d'évitement de congestion. Pour chacune d'elles, nous étudions les problèmes qu'ils posent aux communications des applications MPI selon leur type et les conditions qui aboutissent à ces problèmes.

4.2.1 Description du contrôle de congestion

Lorsque plusieurs sources envoient simultanément à un débit supérieur au lien de sortie d'un équipement réseau, les files d'attente des équipements réseaux se remplissent jusqu'à saturation. Lorsque les files sont pleines, ces équipements jettent le surplus de paquets. C'est ce qu'on appelle le phénomène de congestion.

Afin de résoudre ce problème, TCP utilise une fenêtre de congestion qui détermine la quantité maximale de paquets en vol c.-à-d. la quantité de paquets non acquittées. Cette fenêtre a pour but de déterminer dynamiquement la quantité effective de paquets que le réseau est capable d'absorber. De ce fait, elle contribue à limiter le débit d'émission de manière à éviter les pertes.

La fenêtre de congestion évolue au cours du temps en fonction des informations sur le réseau qui sont rapportées par les acquittements (ACK). La figure 4.2 montre l'évolution de la fenêtre de congestion de TCP New Reno [FHG04] qui est la version standard de TCP. Tout d'abord, TCP utilise un mécanisme de démarrage lent (*slowstart*) de manière à déterminer la bande passante disponible sur le lien. La fenêtre de congestion est initialement fixée à deux paquets, puis elle est augmentée exponentiellement à chaque réception d'un ACK, jusqu'à ce qu'une perte soit détectée ou que la taille de la fenêtre de congestion ait atteint un certain seuil (*slowstart threshold*). TCP quitte alors la phase de slowstart pour passer en mode d'évitement de congestion et divise par deux la fenêtre de congestion.

Durant cette phase, la fenêtre de congestion est incrémentée d'un paramètre α par RTT. α est égal à 1 dans TCP New Reno.

$$cwnd \leftarrow cwnd + \alpha \tag{4.1}$$

Quand une perte est détectée (par trois ACK identiques ou par l'expiration du délai de retransmission (RTO) comme décrit en annexe A.2), la fenêtre de congestion est diminuée en

fonction de β qui est égale à 0,5 dans TCP New Reno. Le RTO est calculé en fonction du RTT ($200\,ms$ + RTT dans Linux).

$$cwnd \leftarrow cwnd - \beta * cwnd \tag{4.2}$$

Enfin, lorsque qu'aucune donnée n'est transmise pendant un certain temps (*idle time*, équivalent au délai de retransmission) ou lorsqu'une perte est détectée par l'expiration du délai de retransmission, TCP recommence à émettre par une phase de démarrage lent.

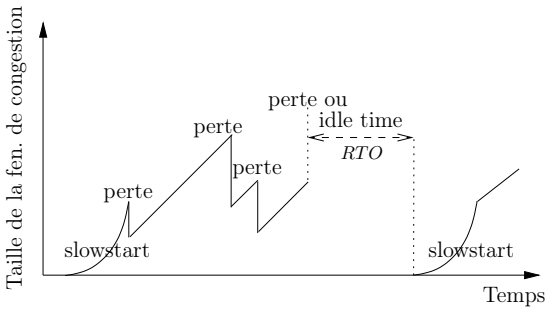


FIG. 4.2 – Évolution de la fenêtre de congestion de TCP New Reno.

4.2.2 Application MPI sans contrôle de congestion

Le contrôle de congestion permet d'émettre un flux à un débit au plus proche de la bande passante disponible sur le lien. Comme le trafic des applications MPI étudiées n'est pas suffisant pour congestionner le lien longue distance, il est envisageable de supprimer le contrôle de congestion en l'absence de goulot d'étranglement sur le WAN, il devrait y avoir peu de pertes. Nous espérons ainsi émettre plus rapidement en n'étant pas limité par la fenêtre de congestion. Nous avons donc étudié si le contrôle de congestion était nécessaire pour l'exécution d'applications MPI dans une grille.

Nos expériences ont porté sur l'exécution des NPB sur des noeuds sur lesquels le contrôle de congestion a été désactivé. Ceci est réalisé en utilisant un module de TCP modifié [SG09] qui force la fenêtre de congestion à une taille qui ne limite pas les émissions sur des liens $1\,Gbit/s$. Nous utilisons 8 noeuds sur chaque site, reliés par un WAN à $10\,Gbit/s$ et avec $11.6ms$ de latence.

La figure 4.3 présente le temps d'exécution sans contrôle de congestion normalisé par rapport au temps où il est activé. Toutes les applications sont impactées par l'absence de contrôle de congestion exceptés MG et CG. Les performances de IS sont catastrophiques car tous les processus communiquent de manière synchrone et dans un temps très limité. Cette chute de performance est due à l'augmentation des pertes et des retransmissions lorsque l'on supprime le contrôle de congestion. En effet, si plusieurs noeuds communiquent vers le même noeud distant, l'interface ne pourra pas traiter tous les paquets reçus.

Le contrôle de congestion est donc nécessaire pour réguler les congestion locales même si le WAN ne constitue pas un goulot d'étranglement puisqu'il offre une bande passante supérieure au débit des noeuds qui communiquent dessus. Nous allons donc évaluer en détail l'influence de chacune des phases du contrôle de congestion de TCP sur les communications MPI.

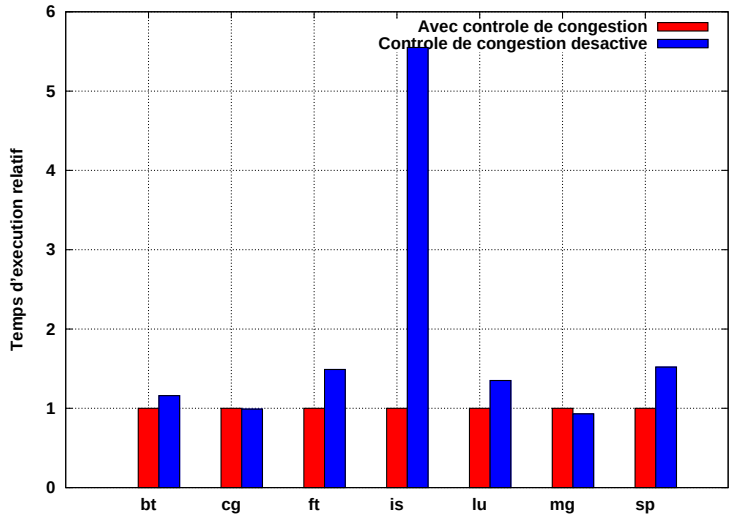


FIG. 4.3 – Temps d'exécution des NPB sans contrôle de congestion relatif au cas avec contrôle de congestion.

4.2.3 Impact du démarrage lent

Dans cette section, nous analysons l'impact du démarrage lent (*slowstart*) qui permet de déterminer la bande passante disponible sur un lien au début d'une communication. Ce mécanisme, décrit en annexe 4.2.1, positionne la fenêtre de congestion à 2 paquets puis la fait croître de manière exponentielle à chaque réception d'un ACK. On quitte cette phase dès qu'une perte est détectée. Le démarrage lent intervient d'une part pour les premières données émises sur une connexion et d'autre part lorsque aucune donnée n'a été transmise pendant un certain laps de temps (idle time) ou lorsqu'une perte est détectée.

4.2.3.1 Impact sur un simple pingpong

Le pingpong est une application qui émet uniquement des données entre deux noeuds et ne fait pas de calcul. Le premier processus émet un message MPI qui est reçu par le destinataire puis retransmis vers l'émetteur. Elle est donc appropriée pour évaluer les communications MPI point à point.

La courbe 4.4 a été obtenue grâce aux outils InstrAppli et `tcp_probe` (présentés en section 3.1.3) et présente le résultat d'un pingpong MPI effectué entre deux noeuds reliés par des liens avec 10.3 ms de latence. La bande passante du lien longue distance est de 1 $Gbit/s$ et ne consitue pas un goulot d'étranglement. Dans cette expérience, la taille des données transférées est de 1 Mio et nous effectuons 100 aller-retours.

Les croix représentent la taille des *writes*, la ligne rouge la taille de la fenêtre de congestion et la ligne verte la quantité de données dans le tampon d'envoi de TCP en fonction du temps. Jusqu'à 0.3 s, le message MPI est envoyé de l'émetteur vers le récepteur ; la fenêtre de congestion croît de manière exponentielle. Entre 0.3 et 0.6 s, le message est renvoyé dans l'autre sens, le tampon d'émission est à zéro. Un peu après 0.6 s on constate la première perte qui divise la fenêtre de congestion par deux et met fin au démarrage lent. On constate la grande

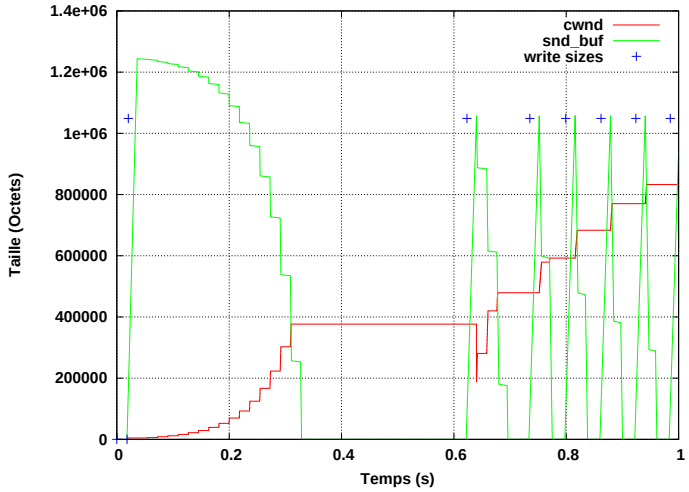


FIG. 4.4 – Influence du démarrage lent sur l'émission de messages MPI de taille 1 Mio.

différence entre le temps aller-retour du premier message (0.7 s) alors que le second met moins de 0.100 ms.

Le démarrage lent est donc à éviter pour les applications MPI, qu'il soit dû à une période d'inactivité ou à une perte qui active fait expirer le délai de retransmission.

4.2.3.2 Impact du démarrage lent après une période d'inactivité

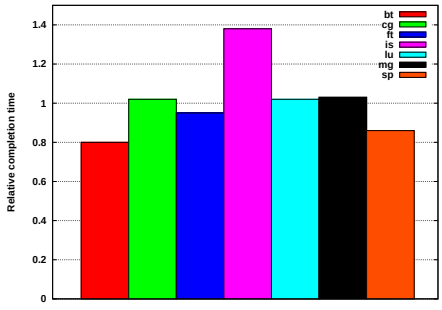
Le démarrage lent après une période d'inactivité est un mécanisme de TCP qui a pour but de re-tester le réseau lorsqu'on n'a pas communiqué sur une connexion pendant un certain temps. Ainsi, lorsque le délai d'inactivité sur un lien est écoulé, on redémarre lentement de manière à éviter d'introduire une quantité de données supérieure à ce que le réseau peut absorber dans son état actuel. Cet algorithme bénéficie à la fois aux flux déjà présents dans le réseau et aux flux qui recommencent à émettre car ils subissent tous moins de pertes.

Cependant, comme nous l'avons observé dans la section 4.2.3.1, le mécanisme de démarrage lent est très coûteux pour les applications MPI.

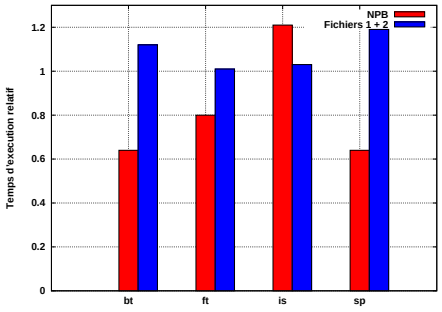
Afin de mettre en évidence le ralentissement induit par le démarrage lent après une période d'inactivité, nous avons exécuté les NPB entre deux grappes en désactivant ce mécanisme. La fenêtre de congestion reste donc à sa valeur précédente même après une période d'inactivité. Le débit du réseau longue distance est à $1\,Gbit/s$. La latence est de $18.1\,ms$. La figure 4.5(a) montre les temps d'exécutions des NPB sans démarrage lent relatif au cas avec démarrage lent après une période d'inactivité. Une valeur égale à 1 indique des performances équivalentes dans les deux alors qu'une valeur inférieure montre une amélioration des performances. On constate que BT, SP et dans une moindre mesure FT, bénéficient de la désactivation de l'algorithme (de l'ordre de 20% pour SP et BT).

En effet, il est nécessaire que les applications communiquent peu souvent pour que le délai d'inactivité expire. Or, comme nous l'avons vu dans la section 3.1.4.3, seules BT, SP, FT et IS entrent dans cette catégorie. Ensuite, les communications ne doivent pas être synchrones sinon l'algorithme est utile : il permet de trouver le juste débit d'envoi afin d'éviter les pertes. De ce fait, IS est très affecté par la désactivation (plus de 40%). FT, par contre, améliore ses

performances de 5% parce que l'algorithme de Alltoall implémenté dans MPICH2 fait en sorte de répartir l'envoi des données entre les processus, ce qui évite la congestion.



(a) Sans trafic concurrent



(b) Avec transfert de fichier en parallèle

FIG. 4.5 – Temps d'exécution des NPB en désactivant le démarrage lent après une période d'inactivité sur le temps sans le désactiver

Cependant, la désactivation de ce mécanisme peut entrainer des pertes conséquentes sur les autres applications qui pourraient s'exécuter en parallèle. S'il est désactivé, après une période d'inactivité, une application MPI recommence à émettre avec un débit correspondant à la fenêtre de congestion de la dernière communication. Si elle n'est pas adaptée à la bande passante actuellement disponible sur le lien, ceci va entrainer des pertes pour les autres applications et contribuer à les ralentir.

Afin d'évaluer l'impact de la désactivation de ce mécanisme sur les applications concurrentes, nous avons effectué deux transferts de fichiers en parallèle de l'application MPI (BT, FT, IS et SP). Un transfert est exécuté dans chaque sens de manière à utiliser totalement la bande passante du lien longue distance. Les fichiers font 40 Gio de manière à ce que leur temps de transfert soit du même ordre de grandeur que les temps d'exécution des applications concernées (excepté IS).

La figure 4.5(b) présente le temps d'exécution des NPB qui peuvent bénéficier de la désactivation avec deux transferts de fichier en parallèle. Les valeurs représentent le temps avec désactivation du démarrage lent sur le temps avec ce mécanisme activé. A coté chaque NPB, nous présentons en vis-à-vis, la somme des temps de transfert des fichiers, pour lesquels le démarrage lent reste activé.

On constate que les applications BT, FT et SP diminuent leur temps d'exécution lorsque que l'on désactive le démarrage lent (de 40%, 20% et 40%). Les temps de transferts de fichiers augmentent de l'ordre de 20% lorsque que les applications BT et SP sont exécutées en parallèle. Comme dans l'expérience précédente, IS est toujours impactée mais le temps de transfert des fichiers n'est pas significatif par rapport au temps d'exécution de l'application car il est grandement supérieur. Par contre, les transferts en parallèle de FT sont quasiement identiques dans les deux cas mais FT améliore ses performances de 20%. Ceci est dû au mode de communication de FT proche d'un envoi avec un flux continu.

Ainsi, la désactivation peut entrainer des pertes de performances conséquentes sur les applications qui s'exécutent en parallèle des applications MPI. Cependant, l'impact de la désactivation est conséquent parce qu'il s'agit d'un transfert de fichier, une application MPI sera moins impactée parce qu'elle ne communique pas en continu. De plus, si on considère uniquement le temps d'exécution total (c'est à dire la somme des temps d'exécution des NPB

et des transferts de fichiers), celui-ci est inférieur dans le cas sans démarrage lent. Ce cas est donc à préférer si on souhaite optimiser globalement les performances des applications de grille.

4.2.4 Impact de la fenêtre de congestion

Comme nous l'avons présenté dans la section 4.2.1, le contrôle de congestion limite la quantité de données en vol sur une connexion. Bien que celle-ci soit nécessaire pour limiter la congestion, elle limite fortement les applications MPI. En effet, si elle n'est pas suffisamment grande le message sera envoyé en plusieurs fois.

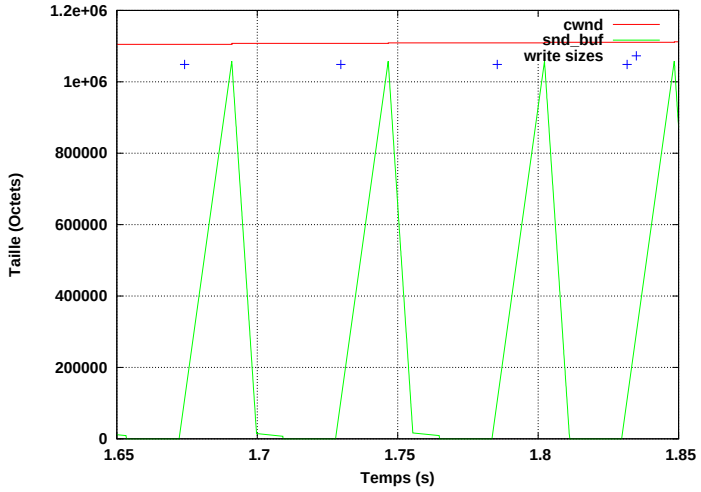


FIG. 4.6 – Délai introduit par la fenêtre de congestion dans l'envoi d'un message MPI.

La figure 4.6 présente l'exécution d'un pingpong dans les mêmes conditions que dans la section 4.2.3.1 entre 1.65 s et 1.85 s. Sur le deuxième pic à 1.75 s, on constate un seuil sur le tampon d'émission. Celui-ci est dû au fait que le message n'est pas envoyé en une fois. En effet, la fenêtre de congestion bloque l'émission du message et ce n'est que lors du retour des ACKs que l'on peut envoyer la dernière partie. Le troisième pic à 1.8 s ne présente pas le même seuil. On constate également que le temps d'émission du message deux est plus longue que celle du message trois.

4.2.5 Performances des variantes de TCP avec les applications MPI

Les sections précédentes ont montré que le contrôle de congestion ne pouvait être supprimé et que certaines applications pouvaient tirer parti d'une désactivation du démarrage lent après une période d'inactivité. Ce mécanisme, même s'il permet de déterminer la taille de la fenêtre de congestion initiale au démarrage de la phase d'évitement de congestion, ne constitue qu'une partie du contrôle de congestion. Nous allons donc nous intéresser à l'évolution de la fenêtre de congestion dans son ensemble (cf fig. 4.2). Nous cherchons à montrer comment celle-ci limite les émissions des messages MPI.

Les sections suivantes présentent d'abord un état de l'art des différentes variantes de TCP disponibles pour les transferts de flux. Ces variantes tentent de résoudre différents problèmes

Variante de TCP	α	β
TCP Reno	1	1/2
BIC-TCP	1 or $bin.search$	1/8
CUBIC	$cub(cwnd, history)$	1/5
HighSpeed TCP	$inc(cwnd)$	$decr(cwnd)$
H-TCP	$f(last_{loss})$	$1 - \frac{RTT_{min}}{RTT_{max}}$
Scalable-TCP	$0.01 * cwnd$	1/8
TCP Illinois	$f(moy(RTT))$	$f(moy(RTT))$

TAB. 4.1 – Constantes AIMD utilisées par certaines des variantes TCP [Gui09].

afférents à TCP en faisant évoluer la fenêtre de congestion différement. Ensuite, nous les évaluons dans le contexte particulier des communications MPI, qui communiquent plutôt de manière discontinue et par rafales.

4.2.5.1 Les différentes variantes de TCP

Nous avons présenté dans la section 4.2.1, l'algorithme de TCP appelé New Reno [FHG04], dans lequel la fenêtre de congestion évolue selon un mécanisme appelé AIMD (Additive Increase Multiplicative Decrease). Différentes études comme [LM97] ont conclu que ce protocole arrivait difficilement à remplir la totalité d'un lien avec un grand produit débit*délai (bande passante * RTT). A la suite de ces recherches, différentes variantes de TCP ont été proposées parmi lesquelles on peut citer BIC [XHR04], CUBIC [RX05], Highspeed [Flo03], Hamilton TCP [SL04] (H-TCP), Scalable [Kel03]. Toutes ces variantes font évoluer différement la fenêtre de congestion en faisant varier les facteurs d'augmentation α et de diminution β (respectivement dans les équations 4.1 et 4.2 de la section 4.2.1). De ce fait, l'efficacité du protocole, l'équité par rapport aux flux dont le RTT est différent et l'équité par rapport aux autres variantes de TCP sont différentes. Le tableau 4.1 issu de la thèse de Romaric Guiller [Gui09] présente les valeurs des paramètres α et β de chacune des distributions disponibles.

D'autres versions comme TCP Compound [TSZS06] ou TCP Illinois [LBS06] proposent d'utiliser également le délai de transmission comme indicateur de congestion. Lorsque le délai de transmission augmente, les paquets restent stockés dans la file d'envoi indiquant que le réseau n'est pas capable d'absorber tout les paquets. TCP Compound et Illinois différent cependant dans leur approche. TCP Compound se sert du délai de transmission pour déterminer une fenêtre de délai qui est ajoutée à la fenêtre de congestion habituelle. TCP Illinois quand à lui utilise les deux informations différement. Comme dans TCP NewReno, la réception d'un ACK va entraîner une augmentation de la fenêtre de congestion ou une diminution si une perte est détectée. Cependant les paramètres d'évolution α et β vont dépendre du délai de transmission. Malheureusement, TCP Compound n'est pas disponible dans les noyaux Linux actuels, elle n'a donc pu être testée.

Nous avons donc évalué le comportement de ces variantes dans un environnement réel, sur la grille Grid5000. Notre première expérience a pour but d'obtenir l'évolution de la fenêtre de congestion. Les valeurs ont été obtenues avec le dispositif expérimental présenté par la figure 4.7 : deux noeuds réalisent l'émission des flux `iperf` depuis un site et une machine réalise la réception de ceux-ci sur un autre site. Le premier flux utilise TCP alors que le second, lancé au bout de 5 secondes utilise UDP avec un débit de 500 $Mbit/s$. La latence était de 18 ms, la bande passante du WAN de 1 $Gbit/s$. Le flux UDP est nécessaire pour réduire le débit

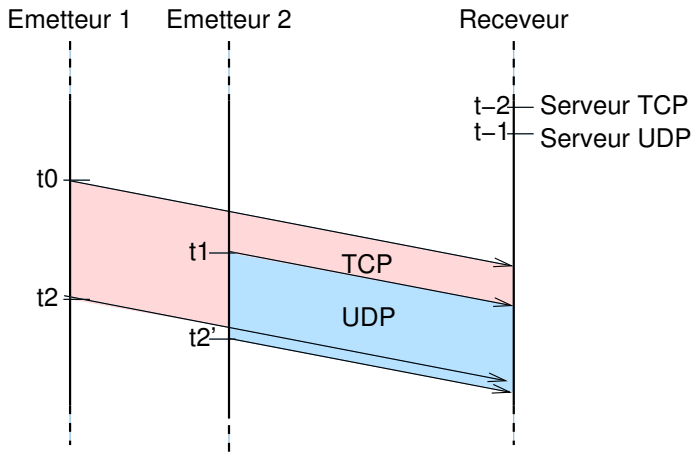


FIG. 4.7 – Dispositif expérimental pour obtenir l'évolution de la fenêtre de congestion de TCP

disponible pour le premier flux et provoquer des pertes qui vont faire évoluer la fenêtre de congestion. Son débit n'est pas dépendant d'une fenêtre de congestion et est donc constant.

La figure 4.8 présente l'évolution des fenêtres de congestion recueuillies avec `tcp_probe`, à partir du moment où on lance le flux UDP (à partir de 5 s). On constate tout d'abord que toutes les variantes subissent l'impact du démarrage lent décrit précédemment. Ensuite, la fenêtre de congestion évolue différemment selon les variantes. La fenêtre de congestion pour BIC lors du démarrage lent grandit à un seuil supérieur aux autres variantes car elle place son seuil de changement de mode (du "*slowstart*" vers l'état stable) plus haut. Reno, CUBIC et Highspeed augmentent leur fenêtre très lentement et contribuent de ce fait à laisser de la place aux autres flux. Les fenêtres de congestion de Scalable et Illinois ont une évolution semblable excepté que la phase d'accroissement de celle de Scalable est globalement plus rapide. Enfin, BIC est la plus aggressive des variantes car sa fenêtre de congestion décroit très peu lors d'une perte et va tenter de ré-augmenter à un seuil plus élevé rapidement.

Cette étude donne seulement une idée de l'évolution de la fenêtre de congestion lorsqu'il y a un seul flux TCP dans le réseau. Lorsque plusieurs flux sont présents, ils interagissent entre eux et font évoluer leurs fenêtres de congestion de manière conjointe.

Aussi, dans [GHK+07], nous avons étudié comment les différentes variantes de TCP se comportaient avec des flux continus (du type transfert de fichiers) sous différentes latences. A la suite de cette étude, nous avons proposé un benchmark pour la comparaison de ces différentes variantes dans [GHVBP07]. Deux cas ont été étudiés : le cas sans congestion et le cas avec congestion. Nous nous sommes intéressés essentiellement à trois métriques : la moyenne des débits et la variance de débit entre les différents flux d'un même lien et l'équité entre les flux. La moyenne des débits nous renseigne sur la capacité des flux à utiliser la totalité de la bande passante disponible. La variance indique si le débit des flux est stable au cours du temps. L'équité précise si tous les flux obtiennent un débit similaire ou s'il y a une grande différence entre eux.

Nous avons effectué l'expérience suivante sur le banc d'essai présenté dans la section 3.7 avec 5 ou 12 noeuds et une bande passante de 10 $Gbit/s$. La latence est émulée à l'aide d'une GtrcNet[1], un équipement réseau qui permet de retarder les paquets pendant un temps déterminé, de 0 à 200 ms. Les flux sont réalisés avec *iperf* entre un couple de noeuds de deux sites

[1] http ://projects.itri.aist.go.jp/gnet/gnet10p3e.html

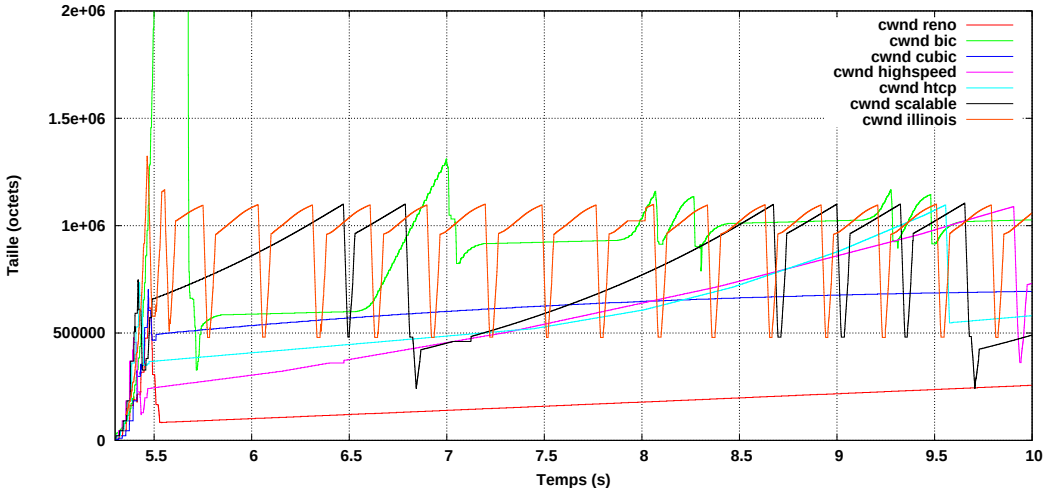


FIG. 4.8 – Evolution de la fenêtre de congestion des différentes variantes de TCP.

différents.

Pour des latences similaires à celle de Grid5000 (inférieures à 20 ms), nous avons conclu que la variante Scalable était meilleure dans le cas sans congestion, alors que CUBIC se distingait pour le cas avec congestion. En effet, ce sont elles qui obtenaient les meilleures performances pour les métriques étudiées. Toutefois, cette étude n'incluait pas Illinois car il n'était pas disponible.

4.2.5.2 Variante de TCP avec une application MPI

Les résultats de la section précédente nous permettent de choisir une variante pour un transfert de fichier par exemple. Cependant, dans le cas d'une application MPI, les contraintes ne sont pas identiques étant donné que les communications n'ont pas lieu en continu et que leurs tailles varient. De ce fait, le débit ou l'équité entre les flux des applications MPI ne peuvent servir à comparer les performances des différentes variantes.

Pourtant le choix d'une variante va avoir un impact sur les performances des applications. En effet, les communications sont limitées par la fenêtre de congestion ; une fenêtre trop petite empêche qu'un message soit envoyé en une seule fois au niveau TCP et ralentit donc l'application. Une première expérience nous a permis de mesurer les temps d'exécution de différentes applications avec différentes variantes de TCP. Nous avons executé les NPB (seulement BT, FT, LU et SP) sur un banc d'essai de 8 noeuds par grappe avec un lien longue distance à $1\,Gbit/s$ et une latence entre les sites de $18ms$. Afin de les comparer à une application qui communique par flux continus, nous avons executé également deux transferts de fichiers de $40\,Gio$, un dans chaque sens. Cette taille de fichier permet d'obtenir un temps de transfert supérieur au temps d'exécution des NPB. Ainsi nous sommes sûrs que le transfert de fichier à un impact sur la totalité de leur communications.

La figure 4.9 présente les temps d'exécution des applications en utilisant les différentes variantes de TCP. Les valeurs pour les tranferts de fichiers sont la moyenne des deux temps de transfert.

On remarque que les transferts de fichiers sont plus rapides si on utilise Highspeed. Ceci est contradictoire avec la section précédente mais ici il n'y a pas de multiplexage des flux. Ainsi, les deux transferts peuvent se dérouler librement sans être affectés par l'autre, excepté par les

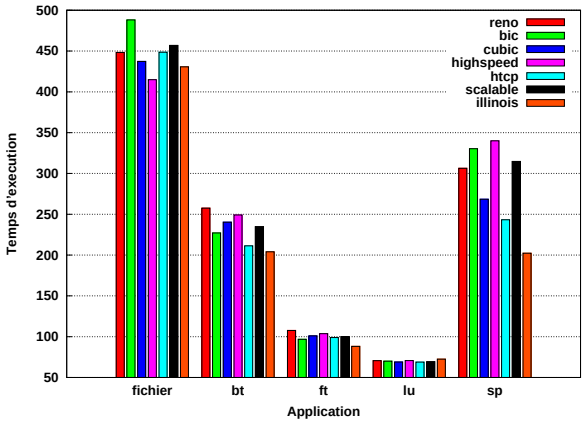


FIG. 4.9 – Exécution d'applications avec différentes variantes de TCP.

acquittements. Le changement de variante a peu d'influence sur le temps d'exécution de FT qui communique par des transferts de gros messages (mais peu en comparaison d'un transfert de fichier). LU est également peu impacté parce qu'il ne crée pas de congestion. En revanche, les temps d'exécution de SP et BT varient d'une variante de TCP à l'autre. La variante Illinois semble la plus adaptée à ce type d'applications suivie de HTCP.

Afin de déterminer comment les variantes de TCP permettent de diminuer le temps d'exécution, nous avons étudié l'évolution de la fenêtre de congestion de SP. A l'aide de InstrAppli, nous avons tracé son évolution au cours du temps ainsi que les `write` sur une des connexions longue distance. Afin d'augmenter la présence de pertes, nous avons ajouté dans chaque sens un flux UDP émettant à $500\,Mbit/s$. Comme précisé précédemment, ces flux n'utilisent pas de fenêtre de congestion et émettent à débit constant. Les résultats pour les différentes variantes sont représentés sur les courbes de la figure 4.10. La fenêtre de congestion de Bic et de Scalable sont basses, de même que celle de Reno pour le dernier tiers des communications. Ceci contribue à ralentir fortement les applications.

Afin d'interpréter les résultats, nous avons cherché des métriques qui pourraient caractériser l'impact de la fenêtre de congestion sur les applications MPI. La taille de la fenêtre de congestion donne une information sur le temps que va mettre un message pour être envoyé. Ainsi, plus elle est grande moins, il faudra de temps pour envoyer un message. Cependant, la moyenne de la taille de fenêtre n'est pas une métrique valide car elle prend en compte les périodes ou l'application ne communique pas. Nous avons donc extrapolé la taille de la fenêtre au moment des `write` et pris la moyenne de ces valeurs. Cependant, si un message est petit et que la fenêtre de congestion est grande, il n'est pas ralenti par cette dernière. Son impact sur ce message est donc nul. L'impact de la fenêtre de congestion sur un message est donc défini comme l'écart de la taille du message par rapport à la taille de la fenêtre de congestion borné par zéro (pas de valeurs négatives). Ces deux valeurs sont présentées dans le tableau 4.2.

Les variantes qui permettent les meilleurs temps d'exécution sont celles dont l'écart par rapport à la fenêtre de congestion est le plus faible (HTCP, Illinois, CUBIC). Celles dont les temps d'exécution sont les plus élevés sont celles dont l'écart est le plus grand comme BIC ou Reno. Pour Scalable, l'écart est grand, la moyenne des fenêtres de congestion faible mais elle parvient à obtenir un bon temps d'exécution. Aussi, pour confirmer ces résultats, il serait

Variante de TCP	Taille moyenne de la fenêtre de congestion lors des `write` (en o)	Moyenne de la différence entre la taille des `write` et la taille de la fenêtre de cong. au moment des `write` (en o)	Temps d'exécution de SP (en s)
TCP Reno	43745	39220	302
BIC-TCP	28842	48167	304
CUBIC	49963	33269	265
HighSpeed TCP	64391	27965	281
H-TCP	54789	30196	250
Scalable-TCP	27261	48952	266
TCP Illinois	59704	28163	248

TAB. 4.2 – Evolution de la fenêtre de congestion lors des `write` issus de l'exécution de SP sur une connexion longue distance.

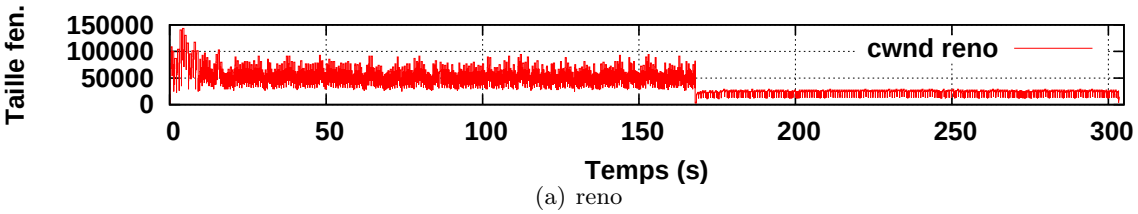
nécessaire de vérifier ces valeurs en tracant l'ensemble des connexions longue distance. Pour Highspeed qui donne un mauvais temps d'exécution, l'inverse se produit avec un écart faible et une moyenne de la fenêtre de congestion élevée. Cependant, notre métrique ne prend en compte que la taille de la fenêtre de congestion au moment de l'envoi. Si celle-ci est trop élevée par rapport au contexte actuel, cela va engendrer des pertes et contribuer à ralentir l'application.

4.2.6 Conclusion sur l'impact du contrôle de congestion

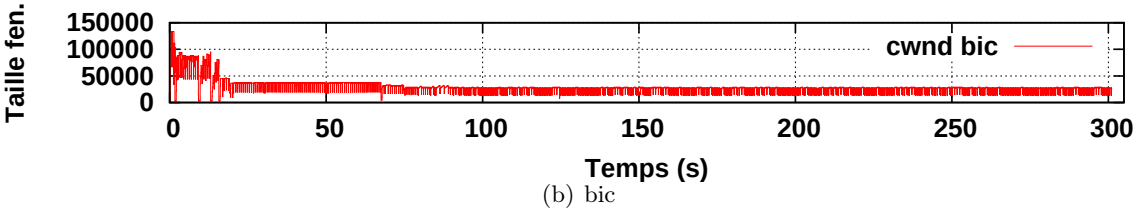
Les différentes variantes étudiées proposent des algorithmes différents pour faire évoluer la fenêtre de congestion. Elles font varier les paramètres d'augmentation α et de diminution β dans le but d'utiliser au mieux le lien longue distance. Les variantes de TCP les plus adaptées pour les flux continus (transferts de fichiers) et les rafales de données (applications MPI) ne sont pas les mêmes. Dans le premier cas, CUBIC semble le plus approprié dans un réseau congestionné en permettant d'obtenir le meilleur débit, tout en garantissant une variance faible et une équité élevée.

Par contre, pour les applications MPI, cette variante n'est pas la plus adaptée même si elle permet d'obtenir de bonnes performances. Dans le cas d'une application qui communique par rafales, Illinois ou HTCP semblent les plus appropriées. Pour Illinois, ceci s'explique par la dynamicité de l'algorithme. Lors d'une perte, la fenêtre de congestion descend bas mais remonte assez rapidement à son niveau précédent en cherchant à ne pas le dépasser. De ce fait, elle laisse la place aux autres flux s'ils en ont besoin mais permet de revenir plus rapidement au niveau optimal de la fenêtre de congestion. Pour HTCP, c'est l'inverse. La fenêtre de congestion diminue très peu sauf si la variation des derniers RTT est grande (ce qui signifie qu'un nouveau flux vient d'arriver). Par contre, son augmentation à partir de ce niveau va être très lente pour ne pas subir d'autres pertes.

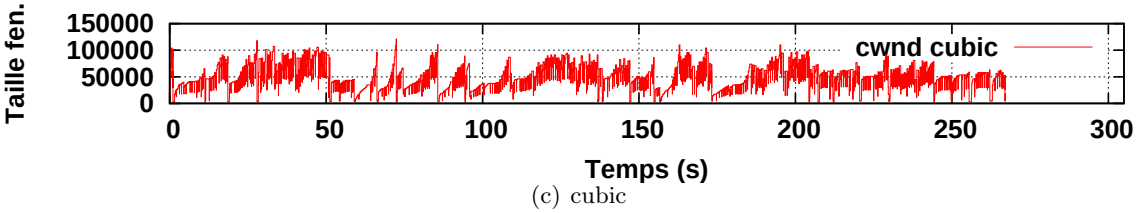
Dans les deux cas, la fenêtre de congestion retrouve assez rapidement un niveau proche du niveau précédent après la perte en cherchant à ne pas le dépasser. Un dépassement trop brutal du niveau optimal va conduire à des pertes qui sont particulièrement coûteuses pour les applications MPI comme nous le montrons dans la section suivante.



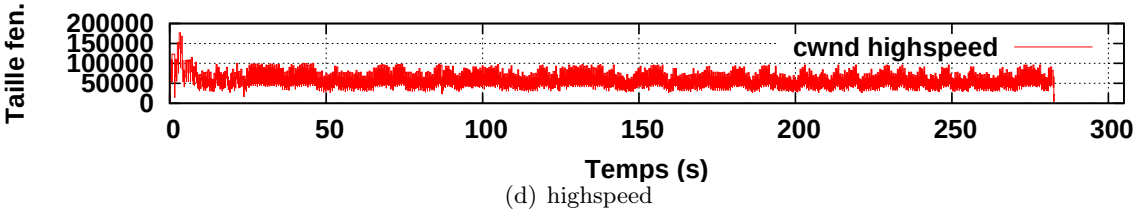
(a) reno



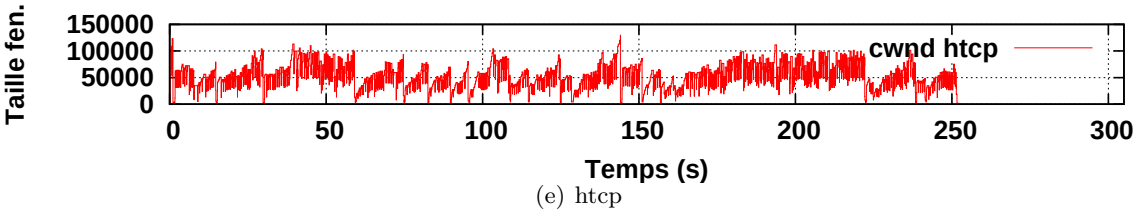
(b) bic



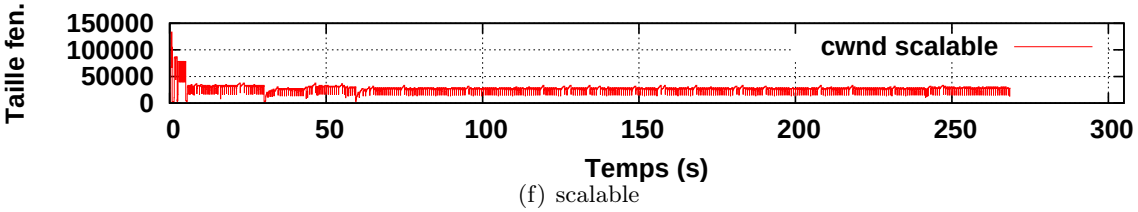
(c) cubic



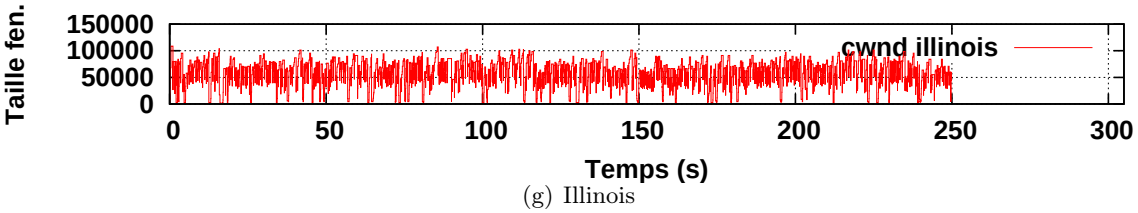
(d) highspeed



(e) htcp



(f) scalable



(g) Illinois

FIG. 4.10 – Evolution des fenêtres de congestion pour SP avec différentes variantes de TCP

4.3 Impact du contrôle de fiabilité

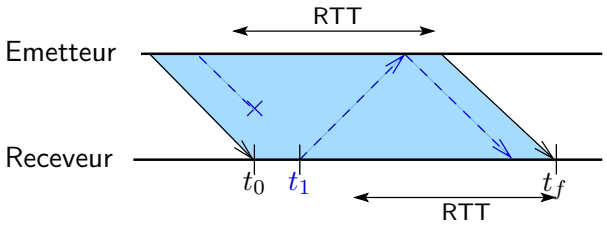
Afin de garantir la fiabilité, c'est à dire la transmission effective des données de bout en bout, TCP utilise un mécanisme de détection de pertes ou d'erreurs et retransmet les segments le cas échéant. Les erreurs sont rares dans les équipements réseaux actuels et sont détectées grâce à la somme de contrôle : la probabilité est de l'ordre de 10^{-9}, ce qui cause une erreur pour 125 Mio transférés ou encore la retransmission d'un paquet sur 80000 environ. De ce fait, les retransmissions sont principalement dues à des pertes, causées par des rejets de paquets dans les équipements réseaux.

Comme il est décrit dans l'annexe A.2, la détection d'une perte peut se faire de deux manières :

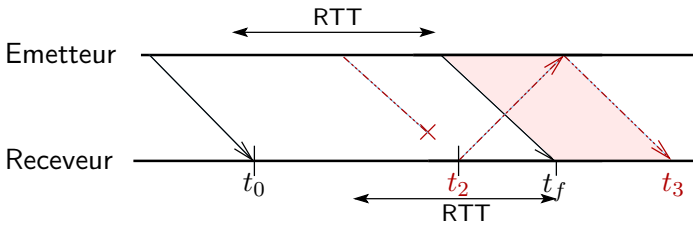
- par la réception de trois ACK identiques (DupACK), dont le délai de réception est de l'ordre d'un RTT (par exemple, celui-ci vaut de 3 à 20 ms dans Grid5000)
- par l'expiration du délai de retransmission (RTO), qui équivaut à 200 ms + RTT.

Comme nous allons le montrer, ces délais sont particulièrement coûteux pour toutes les applications qui envoient de petits messages.

En effet, seules les pertes qui interviennent pendant le dernier RTT affectent réellement une transmission, comme nous allons le montrer sur l'exemple de la figure 4.11(a). Celle-ci montre l'envoi de données sur une connexion TCP si la fenêtre de congestion ne limite pas les émissions. La zone bleue représente la période d'émission des données en temps normal ; la réception à lieu entre t_0 et t_f. Les éléments bleus pointillés de la figure suivent le cheminement d'une perte (matérialisée par la croix) et son recouvrement si elle intervient *avant* le dernier RTT. Le dernier RTT correspond à $t_f - RTT$ c.-à-d un RTT avant la fin de transmission. Dans ce cas, la perte va être détectée puis ré-émise avec les autres données. Le délai introduit par celle-ci est donc négligeable.



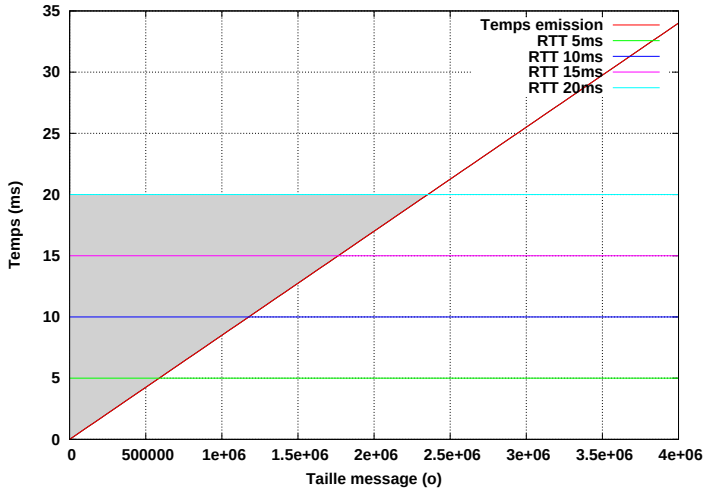
(a) Perte avant le dernier RTT



(b) Perte pendant le dernier RTT

FIG. 4.11 – Transfert de données avec un paquet perdu et retransmis.

Les éléments rouges (lignes discontinues) montrent le cheminement d'une perte qui intervient *pendant* le dernier RTT. La perte est détectée à t_2 puis ré-émise mais entraine le surcoût montré en rouge clair ($t_3 - t_f$). Le délai introduit par les pertes est défini par l'équation 4.3, avec t_f le temps de fin de l'application et t_p le moment où la perte est détectée. Aussi, tout



1

FIG. 4.12 – Temps d'émission d'un message MPI sur un réseau à $1\,Gbit/s$.

envoi dont la durée est inférieure au RTT sera très impacté par une perte.

$$d = \begin{cases} 0 & \text{si } (t_f - t_p) < RTT \\ RTT - (t_f - t_p)) & \text{sinon} \end{cases} \tag{4.3}$$

La courbe 4.12 montre les temps d'envois théoriques d'un message MPI sur un réseau à $1\,Gbit/s$ si la fenêtre de congestion ne bloque pas l'émission. Les droites horizontales pointillées représentent des exemples de RTT dans une grille telle que Grid5000. La zone grisée correspond à la zone où les messages seront ralentis si une perte a lieu pendant leur envoi. Par exemple, si on envoie un message de $1\,Mo$ et que le RTT est supérieur à $8\,ms$, toute perte incluera un délai supplémentaire dans la transmission. Pour tous les RTT inférieurs à $8\,ms$, seule une partie de l'envoi peut être impactée par une perte. Si nous prenons l'exemple des NPB présenté en section 3.1.4.1, la plupart des messages MPI ont une taille inférieure à $500\,Mio$: pour ces messages toute perte sur une connexion avec un RTT supérieur à $4\ ms$ va introduire du délai dans la transmission.

On définit l'impact d'une perte par :

$$I(t_p) = \frac{T}{d} = \begin{cases} 0 & \text{si } (t_f - t_p) < RTT \\ \frac{T}{RTT-(t_f-t_p))} & \text{sinon} \end{cases} \tag{4.4}$$

avec T le temps d'exécution de l'application définit par $T = t_f - t_0$. On constate que le temps perdu du fait d'un retransmission dépend du temps de communication : plus la communication est longue par rapport au RTT, moins son impact est important.

Ainsi, les applications qui transfèrent de petites quantités de données sont particulièrement sensibles aux pertes. Une application MPI est donc plus impactée par une perte qu'un transfert de fichier par exemple.

4.4 Interaction entre le contrôle d'erreur et le contrôle de congestion

Nous avons montré que la fenêtre de congestion pouvait ralentir l'envoi de messages MPI. Cependant, celle-ci est utile pour éviter les pertes, d'autant que comme on vient de le voir les pertes sont particulièrement coûteuses pour ce type d'application. Ce qui introduit réellement du délai en revanche c'est la différence entre la fenêtre de congestion d'une connexion et la fenêtre qu'elle pourrait avoir si elle était adaptée au réseau. Ainsi, les variantes de TCP qui remplissent le mieux le lien sont les plus adaptées.

Dans le cadre des applications MPI, une évolution concave de la fenêtre de congestion est la plus appropriée. En effet, même sans parler de délai de retransmission, les pertes les plus coûteuses interviennent à la fin d'un message. Une évolution concave de la fenêtre de congestion tend à maximiser la probabilité de pertes au début des messages et à les diminuer à la fin de l'émission. Ceci explique pourquoi une variante de TCP comme Illinois est particulièrement adaptée aux applications MPI.

Conclusion

Nous avons étudié dans ce chapitre l'interaction entre les communications de l'application MPI et les transferts au niveau de la couche TCP. Il s'agissait de répondre à la question Q2 : Quels mécanismes de TCP limitent les communications des applications MPI dans un réseau longue distance ?

La première partie étudiait l'impact du contrôle de congestion. Nous avons montré que même si le réseau de la grille était dédié et que les applications MPI communiquaient peu, la désactivation de ce mécanisme provoquait des diminutions de performances sur toutes les applications. Nous avons ensuite montré que le démarrage lent ralentissait particulièrement les applications MPI lors des communications sur la longue distance. Ce mécanisme est également utilisé après une période d'inactivité afin de tester la bande passante actuellement disponible sur un lien. Dans les applications MPI, ces périodes d'inactivité sont nombreuses. Les applications dont les temps entre deux communications est supérieur au délai de redémarrage lent peuvent tirer parti de la désactivation de ce mécanisme. Ainsi, elles sont capables d'envoyer plus rapidement les messages. Cependant, la désactivation de ce mécanisme provoque des pertes sur les autres flux qui s'exécutent en parallèle puisque les applications utilisent une partie plus importante de la bande passante disponible.

Le contrôle de congestion de TCP utilise une fenêtre de congestion qui limite l'envoi des messages MPI. Dans le contexte des transferts de flux continus, différentes variantes ont été proposées pour utiliser au mieux le débit disponible dans les réseaux avec un grand produit débit * délai. Nous avons donc étudié comment les variantes pouvaient permettre une amélioration des temps d'exécution des applications MPI. Les résultats montrent que HTCP ou Illinois sont les variantes les plus adaptées au trafic MPI composé de rafales de données. Après une perte, ces variantes sont celles qui permettent de revenir le plus rapidement à une taille de fenêtre de congestion proche de la taille au moment de la perte. Ainsi elles sont plus dynamiques et sont plus adaptées aux applications MPI.

Dans la seconde partie de ce chapitre, nous avons étudié l'impact du contrôle d'erreur sur les applications MPI. Nous avons montré que l'impact d'une perte était plus important sur les communications courtes. Notre étude montre également que l'impact dépend du RTT : plus le RTT est grand, plus le surcoût lié à la retransmission est important.

D'une manière générale, les mécanismes de TCP dépendent du RTT. La fenêtre de congestion donne une information sur l'état du réseau qui est en retard d'un RTT puisqu'elle dépend des acquittements. Le recouvrement d'erreurs dépend également du RTT et impacte grandement les communications d'une durée inférieure au RTT. Or le temps de transmission des messages MPI est la plupart du temps inférieur au RTT. Ces deux contraintes montrent que la réactivité du protocole TCP n'est pas adaptée aux durées des communications des applications MPI qui sont courtes. Il est donc nécessaire de mettre en place des mécanismes qui permettent de solutionner ce problème.

Chapitre 5

Eclatement des connexions TCP pour les applications MPI

Introduction

Au cours des précédents chapitres, nous avons montré que la grille pouvait être un environnement viable pour l'exécution d'applications MPI. Cependant, les mécanismes de TCP, liés aux caractéristiques des réseaux longue distance, ralentissent les transmissions de messages MPI sur ce type de réseaux. Le contrôle de congestion limite l'émission des messages MPI lorsqu'il utilise une fenêtre de congestion qui n'est pas adaptée aux conditions du réseau. Certaines variantes de TCP permettent de résoudre une partie de ces problèmes mais sont toujours limitées par la faible quantité d'informations disponibles sur l'état du réseau. La connaissance de l'état du réseau est en effet liée à la quantité de données envoyées qui est limitée pour les applications MPI. Nous avons également démontré que le contrôle d'erreur avait un impact important sur les envois de petites tailles. Ainsi, pour ces messages, toute perte se traduit par un surcoût de l'ordre du RTT.

Au regard des problèmes soulevés dans les précédents chapitres, ce chapitre tente de répondre à la question Q3 qui est : comment réduire l'impact des contrôles de TCP sur les communications MPI longue distance ? Pour cela, il convient non seulement de réduire les délais de retransmission mais aussi de limiter la rétention de données par des fenêtres de congestion inadaptées.

Ce chapitre s'appuie sur le fait que les réseaux locaux et longue distance ont des caractéristiques différentes, notamment en terme de latence et de bande passante, ce qui a un impact direct sur les mécanismes de contrôle de TCP. Nous allons donc chercher à différencier les deux types de réseaux, de manière à optimiser spécifiquement les communications sur le longue distance.

Ainsi, notre proposition se base sur un éclatement des connexions TCP en remplaçant une connexion de bout en bout par trois connexions : LAN-LAN, WAN-WAN et LAN-LAN. Cette solution nécessite de placer des mécanismes d'adaptation à l'interface LAN-WAN de chaque site. Dans un premier temps, nous étudions pourquoi l'éclatement des connexions TCP peut être bénéfique pour les applications MPI. Dans une seconde partie, nous proposons une architecture transparente à base de passerelles permettant de réaliser cette optimisation. Enfin, la troisième section évalue l'apport de cette architecture sur les applications MPI.

5.1 Eclatement des connexions TCP

Afin de prendre conscience de la topologie de la grille au niveau de TCP, il est nécessaire de distinguer les connexions LAN des connexions WAN. Ceci peut être réalisé en éclatant une connexion TCP longue distance par trois connexions différentes : une connexion locale, une connexion longue distance, et une connexion locale. Ainsi, il est possible de proposer des optimisations spécifiques sur le réseau longue distance.

5.1.1 Principe

L'éclatement des connexions nécessite l'introduction de passerelles à l'interface entre chacune des connexions. La figure 5.1 illustre le découpage des connexions TCP par l'utilisation de passerelles. Les lignes rouges pointillées sur la figure représentent les connexions dans le cas sans utilisation de passerelles alors que les traits continus verts représentent les connexions avec passerelles. Chaque connexion LAN-WAN-LAN est remplacée par trois connexions LAN-LAN, WAN-WAN, LAN-LAN. De plus, on remplace plusieurs connexions WAN par une seule. $N_{s,n}$ est le noeud n du site s. $P_{s,n,p}$ est le processus p exécuté sur le noeud n du site s. G_s est

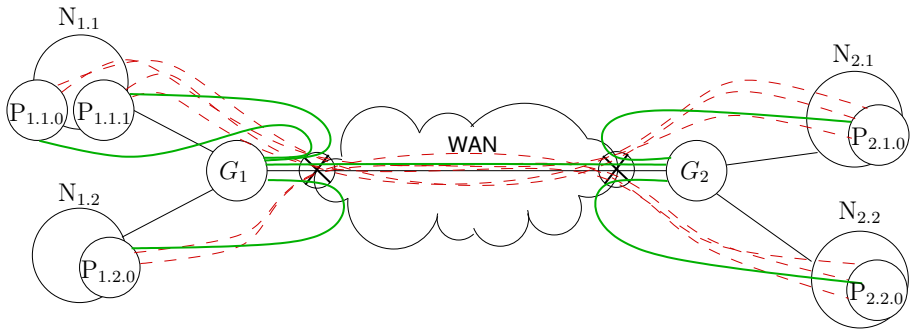


FIG. 5.1 – Eclatement des connexions TCP à l'aide de passerelles.

la passerelle et Sw_s le commutateur du site s. Chaque processus est connecté à la passerelle de son site et les passerelles sont connectées entre elles. Par exemple, au lieu de réaliser une connexion entre les processus $P_{1,1,0}$ et $P_{2,1,1}$, l'éclatement crée une connexion entre $P_{1,1,0}$ et G_1, G_1 et G_2, G_2 et $P_{2,1,1}$.

5.1.2 Optimisations existantes basées sur des passerelles

Les passerelles servent notamment à optimiser les communications dans des réseaux qui n'ont pas les mêmes caractéristiques de bout en bout. Différentes études ont montré que l'exécution de TCP dans des environnements particuliers tels que des liaisons satellites ou sans fil, pouvait engendrer des pertes de performances importantes. Dans ces environnements, il a été prouvé que l'utilisation de passerelles (ou "*proxies*") pouvait améliorer sensiblement les temps de transmission entre le réseau filaire et le réseau dégradé. Ainsi, la RFC3135 [BKG+01] décrit le principe général des PEP (Performance Enhancing Proxies). Ces passerelles permettent d'optimiser une partie d'une connexion point à point qui utilise plusieurs liens aux caractéristiques différentes.

SaTPEP [VKM02] et PEPSal [CC06] proposent d'utiliser un "*proxy*' dans les réseaux à très forte latence tels que les réseaux satellitaires pour permettre de différencier le réseau filaire —rapide— et le réseau satellite —lent et avec une bande passante plus faible—. Ainsi, les communications sur le lien satellite peuvent être optimisées pour ce type de réseau. Le débit s'en trouve amélioré.

Dans les réseaux sans fil, un mécanisme similaire, appelé SplitTCP [KKFT02], permet de distinguer la partie sans fil —soumise à beaucoup de pertes— de la partie filaire.

Enfin, dans les réseaux filaires longue distance, Swany [Swa04] propose de placer des dépôts sur le chemin des données afin de réduire la latence relative subie par chaque paquet. Cette solution est basée sur la manière dont TCP limite l'émission des paquets. Ainsi, lors d'une perte, seule une des connexions du chemin global va subir cette perte et diminuer sa fenêtre de congestion. De plus, la fenêtre de congestion mettra moins de temps à revenir à son niveau précédent puisque sa vitesse d'évolution est inversement proportionnelle au RTT.

Toutes ces approches tentent de réduire les retransmissions sur les liens avec le moins de pertes de manière à réduire la latence globale et ainsi améliorer les performances des transferts.

Nous avons une approche similaire dans le sens où nous éclatons les connexions pour différencier deux réseaux différents : le LAN et le WAN. Cependant, notre contexte est légérement différent des propositions précédentes. Tout d'abord, il se place dans le contexte des réseaux filaires qui subissent peu d'erreur dues au médium de communication. Ensuite, les latences mises en jeu sont très nettement inférieures à celles des réseaux étudiés par Swany [Swa04]. De

plus, les solutions proposées ne proposent pas d'aggréger le trafic comme nous allons le faire sur le réseau longue distance.

5.1.3 Avantages et inconvénients

Cette section évalue les bénéfices que peuvent tirer parti les applications MPI de l'utilisation de passerelles. L'éclatement des connexions présente intrinsèquement les avantages suivants :

- diminution du nombre de connexions et donc de la quantité de mémoire utilisée (du fait des tampons d'émission et de réception) : grâce à l'aggrégation des connexions WAN, le nombre de connexions longue distance devient fonction du nombre de sites, auquel s'ajoutent un nombre de connexions locales égal au nombre de processus.
- diminution des pertes longue distance : comme les passerelles émettent au débit du lien longue distance, la congestion potentielle sur ce lien sera uniquement liée au trafic concurrent.
- fenêtre de congestion plus proche de la capacité réelle du lien longue distance : les passerelles aggrègent tout le trafic MPI sur le lien longue distance. Le trafic résultant sera plus proche des habituels flux gérés par TCP c'est-à-dire des flux continus du type des transfert de fichiers. Ainsi la fenêtre de congestion longue distance sera mise à jour plus souvent et sera de ce fait plus proche de la bande passante réellement disponible sur le lien.
- détection de pertes plus rapide : grâce aux passerelles, situées sur le même site que les noeuds, les ACK sont générés plus rapidement et le temps de détection d'une perte est donc diminué.

A l'inverse, les passerelles présentent l'inconvénient d'augmenter le temps de transmission d'un message. En effet, celles-ci impliquent plusieurs recopies supplémentaires de données qui augmentent la latence logicielle des transferts entre deux processus.

En outre, l'éclatement des connexions, grâce à l'introduction de passerelles, peut aussi permettre des optimisations spécifiques au réseau longue distance :

- Réservation de bande passante ou limitation de débit entre les sites,
- Utilisation de différentes stratégies en fonction de la taille des messages : les petits messages seraient envoyés sur une seule connexion, les gros messages sur plusieurs connexions parallèles pour favoriser le multiplexage TCP,
- Utilisation d'une variante de TCP différente sur le WAN et sur le LAN (par exemple Reno sur le LAN and HighSpeed TCP sur le WAN),
- Traversée de pare-feu : utilisation d'un relais dans une version modifiée [CHP+08, Rez09] d'OpenMPI pour réaliser la connexion par l'intermédiaire d'un proxy entre deux grappes inaccessibles directement,
- Utilisation d'adresses IP privées à l'intérieur d'un site (NAT) : GridMPI[TMK+08].

Les avantages et inconvénients énoncés ci-dessus sont précisés et discutés dans les sections suivantes.

5.1.3.1 Surcoût des recopies

Chaque passerelle introduit deux surcoûts : un surcoût physique et un surcoût logiciel. Le surcoût physique est dû au chemin supplémentaire pour envoyer les données vers la passerelle. Il correspond au temps d'un aller-retour entre la passerelle et le commutateur commun à la passerelle et au noeud (environ $80\mu s$ sur notre plateforme). Comme le montre la figure 5.2, le surcoût logiciel est dû à l'introduction de recopies supplémentaires sur les passerelles : recopie

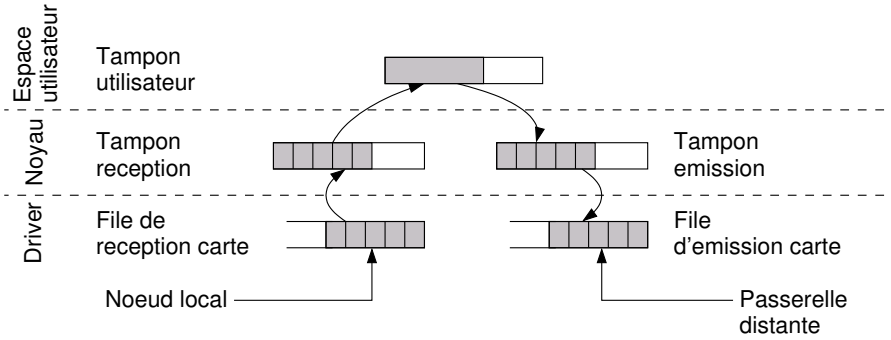


FIG. 5.2 – Recopies dans chaque passerelle.

de la carte vers le tampon de réception de TCP, puis de ce tampon vers l'espace utilisateur (ou sont exécutées les passerelles). Les même recopies ont lieu dans le sens inverse (de l'espace utilisateur vers la carte).

5.1.3.2 Diminution du nombre de connexions

Comme nous pouvons le voir sur la figure 5.1, le nombre de connexions longue distance est réduit. Les connexions d'un noeud vers les noeuds d'un même site ne sont pas prises en compte. Si on pose S, le nombre de sites et P_i, le nombre de processus sur le site i ($P_i = \sum_{s=1}^{N} \sum_{s=1}^{P} P_{i,n,p}$), C_w le nombre de connexions longue distance et C_l le nombre de connexions locales supplémentaires, nous avons :

- dans le cas sans passerelles, pour un noeud, nous avons une connexion vers chacun des noeuds distants : $C_w = \sum_{s=1}^{S-1} P_s * (\sum_{i=s+1}^{S} P_i)$ connexions longue distance.
- dans le cas avec passerelles, seules les passerelles ont une connexion longue distance vers chacun des autres site : $C_w = \sum_{s=1}^{S-1} S-s$ connexions longue distance. Chaque noeud ajoute également une connexion locale vers la passerelle $C_l = \sum_{s=1}^{S} P_s$.

Le nombre de connexions longue distance dans le premier cas augmente donc exponentiellement en fonction du nombre de processus. Dans le second cas, l'augmentation est toujours exponentielle mais en fonction du nombre de sites. Le tableau 5.1 montre des exemples du nombre de connexion dans les deux cas en faisant varier le nombre de sites et le nombre de processus par sites.

Comme nous avons pu le voir dans la section 3.2.1, la taille des tampons d'émission de TCP doit être égale à RTT * bande passante. La réduction du nombre de connexions longue distance va donc diminuer la quantité de mémoire utilisée pour les sockets et rendre éventuellement cette mémoire disponible à l'application. Ceci est d'autant plus important dans un contexte de grille puisque celle-ci augmente à la fois le nombre de machines et à la fois la latence. Le tableau 5.2 présente le gain mémoire sur les noeuds en faisant varier le nombre de sites et le nombre de processus par site. La latence considérée sur le longue distance est de 10 ms et le débit est de 1 Gbit/s. Dans le cas de 8 sites et 100 processus par site, on gagne plus de 800 Mo ce qui correspond à 1/4 de la mémoire des noeuds actuels.

	P		
S	**4**	**100**	**500**
2	16 / 1	10 000 / 1	250 000 / 1
4	96 / 6	60 000 / 6	1 500 000 / 6
8	448 / 28	280 000 / 28	7 000 000 / 28

TAB. 5.1 – Nombre de connexions longue distance avec et sans passerelles

	P		
S	**4**	**100**	**500**
2	4.92	123	615
4	14.68	367	1835
8	32.44	811	4055

TAB. 5.2 – Gain de mémoire sur un noeud par rapport au cas sans passerelle (en Mo)

5.1.3.3 Diminution des pertes sur le réseau longue distance

L'éclatement des connexions TCP contribue à déplacer le goulot d'étranglement du lien longue distance vers l'interface réseau de la passerelle. Cependant, c'est le fait de rapprocher physiquement l'émetteur des acquittements qui va rendre la détection des pertes plus réactive. Un mécanisme similaire appelé "backpressure" a été développé au niveau Ethernet. A ce niveau, le commutateur indique à la carte d'arrêter d'émettre parce que la file est pleine et que les paquets qu'elle émet sont jetés.

Les avantages de l'éclatement des connexions concernant la détection des pertes sont les suivants et expliqués sur la figure 5.3, sur laquelle nous utilisons les notations décrites précédemment. $P_{1.1.0}$ et $P_{1.2.0}$ sont les processus émetteurs situés sur le premier site, G_1 et G_2 sont les passerelles des sites 1 et 2, $P_{2.1.0}$ et $P_{2.2.0}$ sont les processus récepteurs sur le deuxième site. La zone grise représente le WAN. Les flèches bleues pointillées et noires représentent deux connexions TCP différentes. Enfin, les flèches vertes représentent les connexions WAN-WAN quand on éclate les connexions. Différents cas de figure sont présentés :

– Une perte a lieu sur le LAN (due a un *Alltoall* par exemple) : dans le cas 1 de la figure 5.3, la perte qui a lieu sur le LAN du site 1 est réparée beaucoup plus vite grâce à l'éclatement des connexions. Sans passerelle, si le segment 2 est perdu, l'émetteur $P_{1.1.0}$ attend trois ACKs identiques A1 générés par les segments 1, 3 et 4 sur le noeud $N_{2.1}$, avant de s'apercevoir de la perte et de ré-émettre le segment 2 perdu. Avec les passerelles, l'émetteur $P_{1.1.0}$ attend également trois ACKs dupliqués mais c'est G_1 qui les lui transmet. Ceci ne prend environ que le temps d'un aller-retour jusqu'au commutateur (RTT_l, 0.08 ms) alors que dans le premier cas cette information arrive seulement après environ un RTT longue distance (RTT_w, 10 ms).
  Dans le cas 2a, la perte qui a lieu sur le LAN du site 1 est réparée plus vite encore. En effet, les passerelles transforment un RTO en ACK dupliqués. processus $P_{1.1.0}$ envoie deux messages : le premier est composé des segments 2 et 3 qui sont envoyés vers $P_{2.1.0}$, les segments 2' et 3' contiennent le second message envoyé à $P_{2.2.0}$. Sans passerelle, si le segment 2 est perdu et le segment 3 est le dernier du message, $P_{1.1.0}$ recevra seulement un ACK dupliqué A1 et devra attendre l'expiration du délai de retransmission (RTO, 200 ms

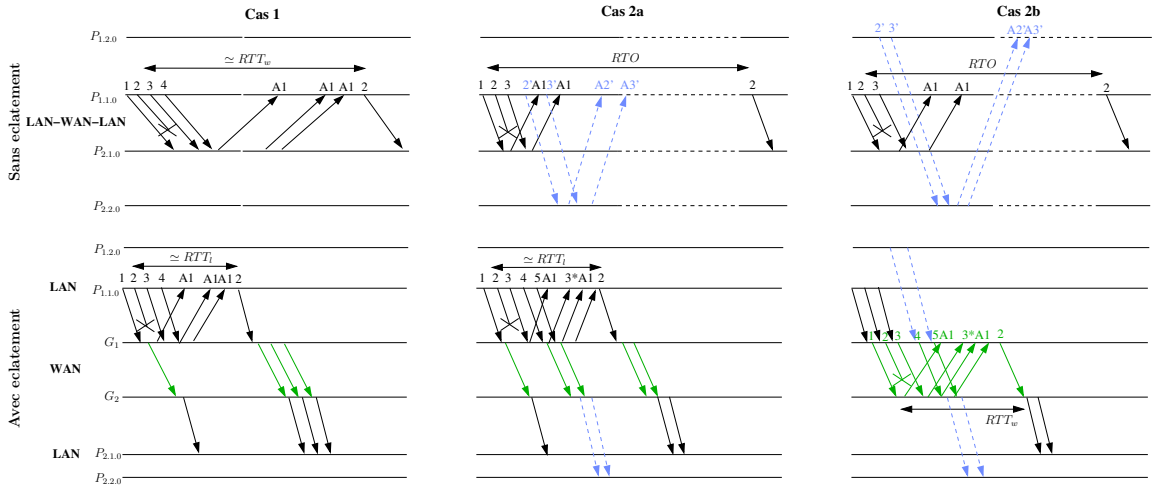


FIG. 5.3 – Analyse du comportement de TCP sans et avec éclatement.

$+ RTT_w$) avant de ré-émettre le segment 2 perdu. Par contre, dans le cas avec passerelles, les segments 4 et 5, correspondants au 2' et 3' du deuxième message à destination de $P_{2.2.0}$, sont envoyés sur la même connexion que les segments 2 et 3 car ils vont tous vers un autre site. Ils contribuent donc à ce que G_1 envoie les ACK nécessaires à la détection de la perte, ce qui est effectué par $P_{1.1.0}$, après qu'il se soit écoulé environ un RTT local (RTT_l).

– Une perte a lieu sur le WAN (due au trafic concurrent) : dans le cas 2b, $P_{1.1.0}$ envoie les segments 2 et 3 vers $P_{2.1.0}$ pendant que $P_{1.2.0}$ envoie les segments 2' et 3' vers $P_{2.2.0}$. Le segment 2 est perdu. Dans sans éclatement des connexions, $P_{1.1.0}$ attend l'expiration du délai de retransmission pour retransmettre le segment perdu. En revanche, avec l'éclatement, les segments 3, 2' et 3' sont aggrégés sur la même connexion entre G_1 et G_2. Lors de leur réception par G_2 comme le segment 2 a été perdu, trois ACK dupliqués sont reçus par G_1 qui peut retransmettre le segment 2 au bout d'environ un RTT longue distance (RTT_w). Cependant, dans le cas avec passerelles, à la réception des segments 3, 2' et 3', G_2 envoie un ACK. $P_{1.1.0}$ retransmet donc le segment après le temps d'un aller-retour longue distance (RTT_w).

L'éclatement des connexions évite l'expiration du délai de retransmission sur les liens locaux et longue distance. Il permet également une réaction plus rapide aux pertes qui ont lieu à l'intérieur d'un site grâce au retour plus rapide des acquittements. De ce fait, il réduit le temps d'attente des applications MPI lié aux pertes et améliore leurs performances d'exécution.

5.1.3.4 Meilleure adaptation de la fenêtre de congestion sur la connexion longue distance

L'éclatement des connexions a également un impact sur la fenêtre de congestion. En effet, comme les passerelles aggrègent tout le trafic MPI provenant du réseau local sur le lien longue distance, elles transmettent plus de données qu'un unique processus. De ce fait, elles sondent le lien plus régulièrement, ce qui permet une évolution et mise à jour plus régulière de la fenêtre de congestion. Celle-ci est donc plus proche de la bande passante réellement disponible. De plus, comme nous l'avons montré dans le chapitre 4, si une application communique peu souvent, elle subit l'impact du démarrage lent après chaque période d'inactivité (c.-à-d. sans émission).

Dans ce cas, les passerelles contribuent à ne pas retourner en démarrage lent. Les messages envoyés par d'autres processus permettent de réinitialiser l'horloge de redémarrage lent en émettant sur ce lien.

5.2 MPI5000 : Implémentation d'une architecture mettant en œuvre l'éclatement des connexions

La section précédente présentait les avantages et les inconvénients d'un éclatement des connexions TCP. Nous avons vu que ce mécanisme pouvait être intéressant pour les applications MPI parce qu'il peut permettre de résoudre une partie des problèmes soulevés dans le chapitre 4 : pour les communications longue distance grâce à l'aggrégation de tout le trafic local sur une seule connexion, il régule le trafic en rafales des applications MPI afin de le rendre plus proche d'un trafic sous forme de flux continus, mieux adapté à TCP.

Cette section présente, MPI5000, l'architecture à base de passerelles que nous avons développée pour réaliser l'éclatement des connexions TCP. Cette architecture est totalement transparente du point de vue de l'application et ne nécessite aucune modification du code de l'application ou de l'implémentation. De plus, elle est compatible avec n'importe quelle implémentation MPI. Seul un ajout de trois lignes dans le script de lancement des processus MPI est nécessaire.

La figure 5.4 présente une vue d'ensemble de MPI5000, qui est basé sur deux composants principaux : une librairie et des démons qui tournent sur les passerelles. La ligne en pointillés rouges représente le chemin des données d'un noeud vers l'autre dans le cas sans passerelle. La ligne continue verte représente le cas avec passerelles. A ces deux composants principaux, s'ajoutent des scripts qui permettent d'effectuer le lancement des passerelles et de créer les fichiers nécessaire au fonctionnement de la librairie.

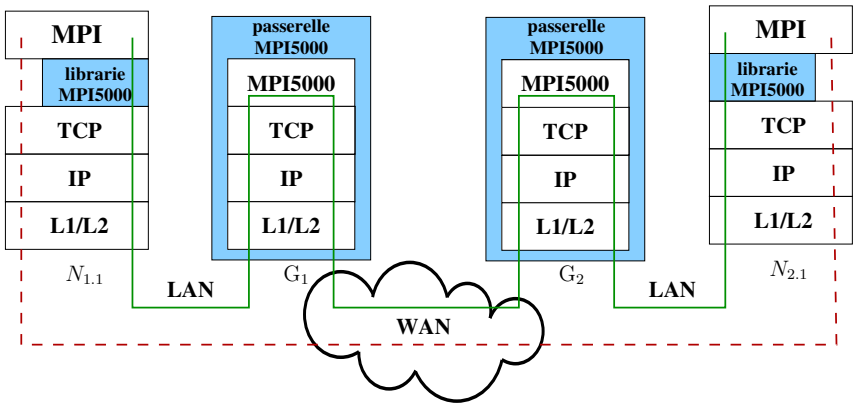


FIG. 5.4 – Traversée des différentes couches de communication avec MPI5000 (en vert) et sans (en pointillés rouge).

Dans la suite de cette section, nous décrivons les éléments de notre architecture : la librairie, les passerelles, l'entête et le routage des messages et les étapes pour utiliser MPI5000.

5.2.1 La librairie MPI5000

Cette librairie est placée en dessous de l'implémentation MPI de manière à permettre son utilisation avec toutes les implémentations existantes. La librairie intercepte les appels aux

fonctions de l'API socket (`bind, accept, connect, poll, write, writev, read, readv, close`) afin de réaliser l'éclatement des connexions. En d'autres termes, nous forçons l'implémentation MPI à appeler nos fonctions au lieu des fonctions de la librairie standard, afin de les adapter à notre architecture. Cette librairie ne modifie que les connexions distantes, les autres connexions restent inchangées. Comme nous le verrons par la suite, la différentiation entre une connexion locale au site et une connexion distante est faite par un fichier qui détermine si le destinataire est situé sur le même site ou non.

Dans le cas sans passerelles, les appels aux fonctions `connect` et `close` ont une action sur le processus distant. Un `connect` associe un descripteur de fichier à un adresse IP et un port, et libère le `accept` distant en attente. Le `close` enlève le descripteur de fichier de la liste des descripteurs ouverts et il indique au processus distant que cette connexion est fermée. Lors de l'éclatement des connexions, il faut donc propager les appels de fonctions du site local vers le noeud du site distant. Nous transférons pour cela des messages de contrôle correspondants au `connect` et au `close` à l'aide du drapeau placé dans l'entête.

Les algorithmes de la librairie sont détaillés en annexe C. Lorsqu'un `bind()` est intercepté, la librairie réalise la connexion à la passerelle, effectue l'appel classique et enregistre le port et le numéro de socket sur lequel il est effectué. En prenant exemple sur la figure 5.1, si on se place sur le processus $P_{1.1.0}$ effectue un `connect()` vers $P_{2.1.0}$, la librairie trouve l'identifiant de la passerelle du site distant (à l'aide du fichier de passerelles) et lui envoie un message de contrôle. Le `accept()` correspondant du processus $P_{2.1.0}$, attend juste le `connect()` adéquat, provenant de la passerelle G_2, qui viendra libérer celui-ci (comme il sera expliqué dans la section suivante). Lors d'un `write`, la librairie identifie le noeud distant (à l'aide du fichier de noeud) et transmet l'entête et les données à la passerelle. Lors d'un `read`, la librairie effectue le `read` sur l'unique connexion distante vers la passerelle. Enfin, le `poll` est nécessaire pour indiquer à l'implémentation MPI que des données sont arrivées sur une des sockets qu'elle scrute.

5.2.2 Les passerelles

5.2.2.1 Fonctionnement

L'algorithme des programmes des passerelles comporte trois étapes : les passerelles se connectent entre elles, puis elle attendent les connexions provenant des noeuds et enfin, elles retransmettent le trafic qu'elles recoivent des noeuds vers la passerelle distante adéquate ou bien d'une autre passerelle vers le bon noeud. Les programmes exécutés sur les passerelles de MPI5000 s'exécutent en arrière plan (sous forme de démon).

Nous utilisons une unique connexion pour communiquer sur le WAN. Celle-ci remplace les n connexions qui auraient été nécessaires sans l'utilisation de passerelles. Quand un programme passerelle reçoit un message de contrôle correspondant à un `connect`, elle réalise un vrai `connect` vers le noeud adéquat. Ce mécanisme est nécessaire pour relacher l'`accept` en attente sur le noeud distant. Si on prend l'exemple de la figure 5.1 ou 5.4, lorsque la passerelle G_2 reçoit un message de contrôle lui indiquant une connexion à destination du processus $P_{2.1.0}$, elle se connecte à ce processus. Ceci permet que l'`accept` qui attendait cette connexion puisse rendre la main. Après que l'`accept` a été relaché, la socket de connexion est close de manière à ce qu'il ne reste qu'une unique connexion entre la passerelle et le noeud : celle qui a été créée lors du `bind`. De cette manière, nous diminuons l'impact des sockets sur la consommation de ressources notamment en quantité de mémoire utilisée comme nous l'avons vu précédemment en section 5.1.3.2. De la même manière, nous propageons le `close`. Les algorithmes des passerelles sont

	Temps d'exécution en μs
Deux `read`, deux `write`	12069
Un `read`, deux `write`	12068
Deux `read`, un `write`	12052
Un `read`, un `write`	12051

TAB. 5.3 – Temps d'exécution d'un pingpong entre deux sites, la latence est de 11.6 ms.

également décrit en annexe C.

Les passerelles peuvent contribuer à créer des inter-blocages. Si on prend l'exemple d'un transfert d'un gros message de $N_{1.1}$ vers $N_{2.1}$ via G_1 puis G_2. Si l'application ne lit pas les données transmises sur $N_{2.1}$, le tampon de réception de celui-ci va se remplir. De ce fait, G_2 sera bloquée puis G_1 puis $N_{1.1}$. Ceci pose problème si $N_{2.1}$ émet également des messages qui ne seront pas retransmis par G_2 puisqu'elle attend de pouvoir écrire des données. Il a donc été nécessaire de mettre en place des *threads* sur les passerelles. Ainsi, elles peuvent effectuer les retransmissions dans les deux sens simultanément : un *thread* s'occupe des retransmissions des noeuds vers la passerelle distante, l'autre retransmet les données qui arrivent d'une autre passerelle à destination des noeuds.

5.2.2.2 Paramétrage optimal de la taille des lectures/écritures

Nous avons étudié l'impact de la transmission de l'entête ajouté par MPI5000. Celle-ci peut être transmise et lu en même temps que les données ou bien séparément. Nous nous sommes tout d'abord intéressés à savoir si le nombre de `write` et de `read` impactait les applications. Nous avons pour cela exécuté un simple pingpong qui écrivait une quantité de 32 o (entête) plus 1 Kio (données). Ces messages ont été écrits/lu, soit en une seule fois (à l'aide d'un `writev/readv`), soit l'un à la suite de l'autre (grâce à deux `write/read`). Les résultats sont regroupés dans le tableau 5.3 et montrent clairement que le nombre de `read` a peu d'impact mais que le nombre de `write` influe sur les performances. L'entête est donc écrit en même temps que les données, à l'aide d'un `writev` afin de réduire le temps de retransmission.

La taille des blocs de données lues et écrites par la passerelle est configurable. En effet, pour garantir un chevauchement de la lecture et de l'écriture des données, il est préférable de ne pas lire toutes les données en une seule fois. Nous nous sommes donc intéressé à la configuration optimale.

Nous avons émis un flux entre deux noeuds en faisant varier la taille des blocs de données lus et écrits par les passerelles. La figure 5.5 montre le débit obtenu en fonction de la taille des `write` avec MPI5000. On constate que le débit optimal est obtenu pour des blocs de 128 kio. C'est donc cette valeur que nous avons conservé par la suite pour nos évaluations.

5.2.3 Description du protocole et du routage

L'entête MPI n'étant pas standardisée, nous sommes obligés d'ajouter une entête, propre à MPI5000, pour garantir la transparence par rapport à l'implémentation. Elle contient les informations nécessaires pour réaliser la redirection des connexions des noeuds vers les passerelles et entre les passerelles.

La figure 5.6 montre l'entête de MPI5000. Elle contient quatre champs :

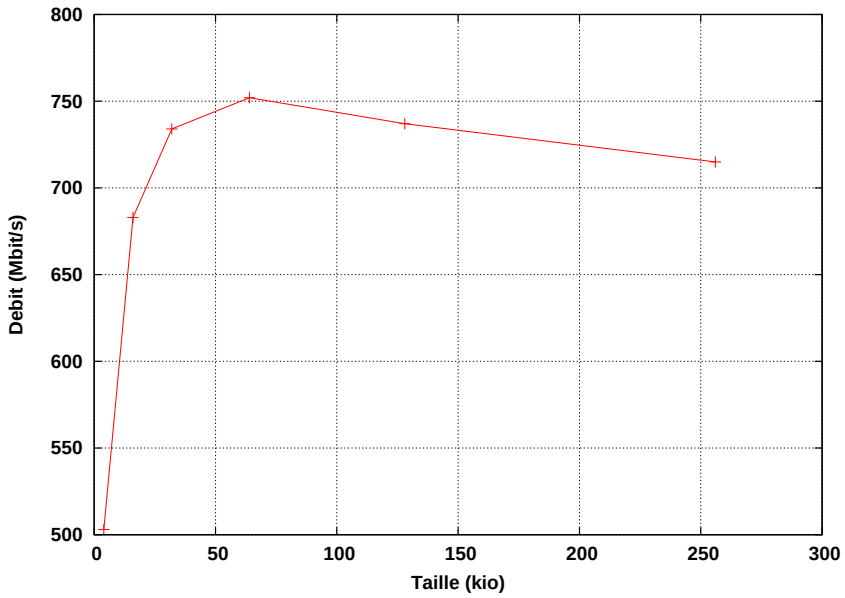


FIG. 5.5 – Débit d'un transfert de fichier en fonction de la taille des `write` avec MPI5000.

- la destination permet de connaître le site et le noeud destinataire,
- la source est nécessaire pour faire la correspondance entre notre numérotation et la socket sur laquelle va lire le processus MPI,
- le flag permet d'envoyer des messages de contrôle à une autre passerelle de manière à réaliser un `connect` ou un `close` distant,
- la taille permet de savoir la quantité de données transmises à la suite de cette entête.

Une source ou une destination est décrite par trois chiffres : s, n, p. Les numéros de site et de processus sont stockés sur 1 o. Le numéro du noeud est stocké sur 2 o. De cette manière, nous pouvons gérer jusqu'à 256 sites, 65536 noeuds par site et 256 processus par noeud.

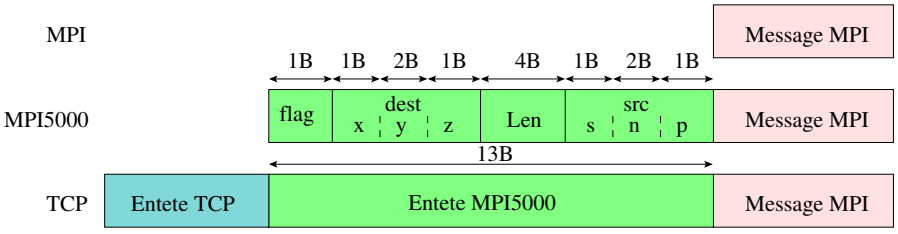


FIG. 5.6 – Entête MPI5000.

5.2.4 Etapes pour un lancement transparent de MPI5000

Cette section décrit brièvement les étapes nécessaires pour utiliser MPI5000 sur une grille telle que Grid5000. En temps normal (c.-à-d. sans MPI5000), le lancement d'une application MPI s'effectue avec MPICH par la commande suivante :

```
mpirun -machinefile mon_fichier -np nb_process mon_appli
```

où `mon_fichier` contient la liste des noeuds, `nb_process` le nombre de processus sur lesquels doit s'exécuter l'application, `mon_appli`, l'application MPI. Cette commande est similaire pour la plupart des implémentations. Nous avons gardé cette commande inchangée mais nous avons modifié le script `mpirun` auquel elle fait appel. Cette modification est limitée à l'ajout de trois lignes : une pour créer les fichiers de configuration à l'aide de `mon_fichier`, une pour lancer les démons sur les passerelles, et une pour charger la librairie sur les noeuds. Chacune de ces étapes est décrite ci-dessous.

Création des fichiers de configuration MPI5000 utilise trois fichiers de configuration qui servent à identifier le site sur lequel la librairie ou le démon s'exécute. En effet, notre architecture nécessite d'identifier les points d'émission et de réception des communications. Sur les noeuds, il est nécessaire de connaitre son propre identifiant et l'identifiant du noeud distant. Sur les passerelles, il faut savoir où rediriger un message, donc connaître l'identifiant de la passerelle distante.

Ces fichiers sont créés à partir de la liste des noeuds du fichier passé en paramètre à `mpirun`. Le tableau 5.2.4 présente un exemple de ces fichiers.

(a) fichier_sites

Nom du site	Identifiant du site
lyon	1
rennes	2

(b) fichier_passerelles

Identifiant passerelle	IP locale	IP distante	Port local	Port distant
1	10.69.1.6	10.69.1.6	10000	30000
2	172.28.54.8	172.28.54.8	10000	30000

(c) fichier_noeuds

Nom du noeud	IP du noeud	Identifiant du noeud
sagittaire-40.lyon.grid5000.fr	10.69.1.40	1.1
sagittaire-41.lyon.grid5000.fr	10.69.1.41	1.2
griffon-30.nancy.grid5000.fr	172.28.54.30	2.1
griffon-31.nancy.grid5000.fr	172.28.54.31	2.2

Les identifiants (site, passerelle et noeud) sont des numéros attribués pour chacun des éléments, et reprennent les notation précédentes. L'identifiant du site est S_s, celui de la passerelle G_s. L'identifiant du noeud est $N_{s,n}$.

La répartition des noeuds par sites est effectuée par le nommage des noeuds, puisque dans Grid5000 ceux-ci sont nommés par <`cluster`>–<`numéro`>.<`site`>. Cependant, dans une grille différente avec une autre convention de nommage, on peut envisager de l'effectuer autrement, par exemple grâce à un service qui donne le site en fonction d'un noeud ou d'une adresse IP.

Le fichier de sites sert à connaitre le nombre de sites et leur identifiant. Il est créé en

effectuant une recherche sur les différents sites utilisés par les noeuds. L'identifiant du noeud est attribué dans l'ordre du fichier après un tri des noeuds par site.

Le fichier de passerelles identifie les noeuds passerelle et détermine les IP et les ports sur lesquelles elles communiquent. Les IP sont récupérées par un connexion a la machine passerelle, et les ports sont attribués arbitrairement. Les IP locale et distante sont identiques s'il n'y a qu'une interface sur les noeuds. La passerelle d'un site est par défaut le dernier noeud de chaque site. Si l'on veut placer les passerelles sur un noeud qui n'exécute pas de processus MPI (ce qui est préférable pour optimiser les performances), il convient de mettre un noeud supplémentaire par site dans le fichier `mon_fichier`, passé en paramètre au `mpirun`.

Le fichier de noeud fait la correspondance entre le nom d'un noeud et son identifiant.

Lancement des démons sur les passerelles Le lancement des démons est effectué par un script qui se connecte à chacune des passerelles pour lancer le démon approprié. Ce programme prend en paramètre le numéro de la passerelle, le nombre de passerelles et le nombre de processus que la passerelle doit attendre.

Chargement de la librairie sur les noeuds La variable d'environnement LD_PRELOAD permet de charger dynamiquement une librairie avant toute autre librairie que nécessite le programme. C'est ce mécanisme que nous avons utilisé pour charger notre librairie. Dans MPICH2, cette variable est positionnée à l'aide du paramètre `-env`. Dans d'autres implémentations, il faut exporter explicitement cette variable.

5.3 Évaluation des performances de MPI5000

Après avoir montré, dans la section 5.1, les avantages que pouvait apporter l'éclatement des connexions TCP pour l'exécution d'applications MPI dans un réseau longue distance, la section 5.2 présentait MPI5000, une architecture à base de passerelles permettant de mettre en oeuvre ce mécanisme.

Dans cette section, nous évaluons les surcoûts et les gains qu'apporte cette architecture pour l'exécution d'applications MPI, afin de démontrer la validité de notre approche. Comme précédemment, notre évaluation est principalement effectuée sur Grid5000.

5.3.1 Surcoût induit par les passerelles

Nous évaluons tout d'abord le surcoût lié à la redéfinition des primitives socket. Ensuite, nous évaluons le surcoût total lors d'un échange entre deux noeuds (pingpong).

Impact de la redéfinition des primitives Notre librairie est basée sur un détournement des fonctions sockets (`connect`, `accept`, `write`, `read`). Nous avons donc étudié le surcoût induit par ce détournement dans l'appel aux fonctions.

La table 5.4 montre, pour les différentes fonctions détournées, la différence entre le temps avec chargement de la librairie et le temps sans. Le surcoût est négligeable sauf pour la fonction `connect`. En effet, le `connect` des implémentations MPI est non bloquant, il retourne donc quasiement instantanément. Lors du détournement, il nécessite le temps supplémentaire d'un RTT local pour se connecter à la passerelle (cf 5.1.3.1). Le temps pour la fonction `accept` n'a pas pu être mesuré puisque sa terminaison dépend du moment où est effectué le `connect` ce qui varie d'une expérience à l'autre.

Fonction	Surcoût (μs)
connect()	121
accept()	non mesurable
read()	0
write()	1

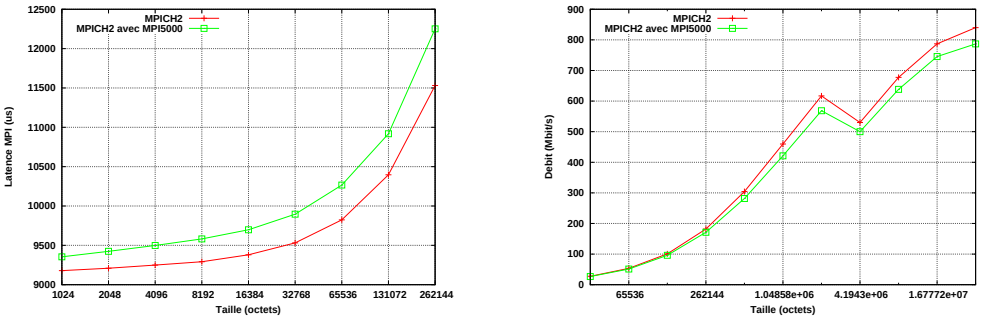
TAB. 5.4 – Impact de l'ajout de la librairie MPI5000 sur les noeuds.

Le surcoût lié à la redéfinition des primitives sockets est donc négligeable ; le `connect` est légèrement plus coûteux mais il n'intervient qu'a l'initialisation de la connexion.

Pingpong Pour évaluer l'impact de MPI5000 sur les communications MPI, nous avons exécuté un pingpong qui permet de mesurer le temps aller-retour d'un message entre deux noeuds. Elle a été réalisée entre deux noeuds reliés par un réseau longue distance avec 18 ms de latence. Les résultats comparent l'exécution de ce pingpong sans et avec MPI5000.

	MPICH2 sans MPI5000	MPICH2 avec MPI5000	Coût
Latence (μs)	9114	9255	141 μs (1.5%)
Débit ($Mbps$)	840	785	-7%

TAB. 5.5 – Latence et débit MPI sans et avec MPI5000 sur la grille.



(a) Latence (b) Bande passante

FIG. 5.7 – Comparaison des performances d'un pingpong MPI sans et avec MPI5000.

Comme le montre le tableau 5.5, la latence est augmentée de 141 μs pour des messages de 1 o. Comme il a été expliqué en section 5.1.3.1, le surcoût introduit par une passerelle correspond au temps d'un aller-retour entre le commutateur et la passerelle, ajouté aux temps de recopies des cartes vers le tampon de réception et de ce tampon vers le tampon utilisateur, et la même chose en sens inverse. Afin de vérifier le surcoût des passerelles de MPI5000 est effectivement limité à ces seuls délais, nous exécutons un pingpong entre deux noeuds d'une même grappe. En effet, le nombre de copies et d'aller-retours est identique au surcoût introduit par nos passerelles. L'exécution de cette expérience donne un temps aller-retour de 142μs, ce qui est similaire au temps mentionné précédemment. Tous les surcoûts de la passerelle sont donc identifiés. Le débit chute de 7% passant de 840 $Mbit/s$ à 785 $Mbits/s$. La figure 5.7

montre que ce surcoût augmente quand la taille des messages croît. Comme pour MPICH2, la courbe présente une chute de débit à $4\,Mio$ qui correspond au changement de mode de MPI (du mode *eager* au mode *rendez-vous*).

Dans toutes nos expériences, nous n'avons utilisé qu'une seule carte à 1 Gbit/s sur les passerelles. Ceci implique que les passerelles divisent leur bande passante par deux si elles doivent émettre ou recevoir dans les deux sens en même temps (vers son site ou vers un autre site et depuis son site ou un autre site) . En effet, même si elles sont capables d'émettre et de recevoir simultanément, elles reçoivent le trafic entrant des autres passerelles mais également le trafic entrant des noeuds du site. Le même problème se produit pour le trafic sortant. Cette limitation matérielle décroit fortement les performances pour les gros messages.

5.3.2 Amélioration des performances grâce aux passerelles

Cette section évalue les performances de notre architecture à l'aide des NPB, en comparant les temps d'exécution de MPICH2 sans MPI5000 avec ceux avec MPI5000. Dans un premier temps, nous évaluons les temps d'exécution que nous mettons en relation avec le nombre d'acquittements dupliqués et d'expiration du délai de retransmission. Dans un second temps, nous étudions le nombre de redémarrages lents dans les deux cas. Ensuite, nous ajoutons du trafic concurrent pour vérifier l'impact sur les points évoqués précédement. Enfin, nous augmentons la taille du problème et le nombre de noeuds pour évaluer le passage à l'échelle de notre architecture.

5.3.2.1 Performances des NPB avec MPI5000

Cette expérience a pour but d'évaluer l'impact de MPI5000 sur les temps d'exécution des NPB. Elle a été menée sur le même banc d'essai que précédemment avec 8 noeuds par site, reliés par un réseau à $1\,Gbit/s$ et une latence de $18ms$. Les passerelles n'ont qu'une seule interface à $1\,Gbit/s$.

Nous avons exécuté les NPB plusieurs fois et prenons le temps d'exécution le plus faible pour s'abstraire de l'utilisation potentielle du réseau longue distance par d'autres utilisateurs. La figure 5.8 montre le temps d'exécution de chaque NPB à l'aide de MPICH2 avec MPI5000 normalisé au temps de MPICH2 sans MPI5000. Un temps relatif inférieur à 1 montre une diminution des temps d'exécution, un temps supérieur à 1 une augmentation des temps d'exécution. MPI5000 permet d'améliorer les performances de BT, CG et SP mais diminue celles de FT, IS et dans une moindre mesure LU et MG.

FT et IS utilisent communiquent de manière synchrone avec de gros messages (cf. 3.1.4.3). L'utilisation d'une seule interface sur les passerelles justifie leurs mauvaises performances, comme nous l'avons expliqué dans la section précédente. Pour confirmer cette idée, nous avons effectué un *Alltoall* avec des messages de 2 Mio avec et sans MPI5000. Le temps d'exécution relatif est de 2.74. Comme le montre la figure 5.8 le surcoût est de 2.86, ce qui signifierait donc que FT n'effectue que des communications (ce qui n'est pas le cas). Le surcoût dû au *Alltoall* n'est donc pas le seul point problématique. Les mêmes observations peuvent être faites avec IS mais les tailles des messages étant plus petites, le surcoût est moindre. Le surcoût des passerelles est élevé si elles ne peuvent retransmettre les données suffisament rapidement car elles deviennent un goulot d'étranglement.

Afin d'analyser les performances des autres NPB, le tableau 5.6 présente le nombre de détection de pertes (DupACK et RTO), obtenus avec le patch Web100 [MHR03]. Ceux-ci sont capturés sur toutes les connexions longue distance des noeuds dans le cas sans MPI5000. Dans

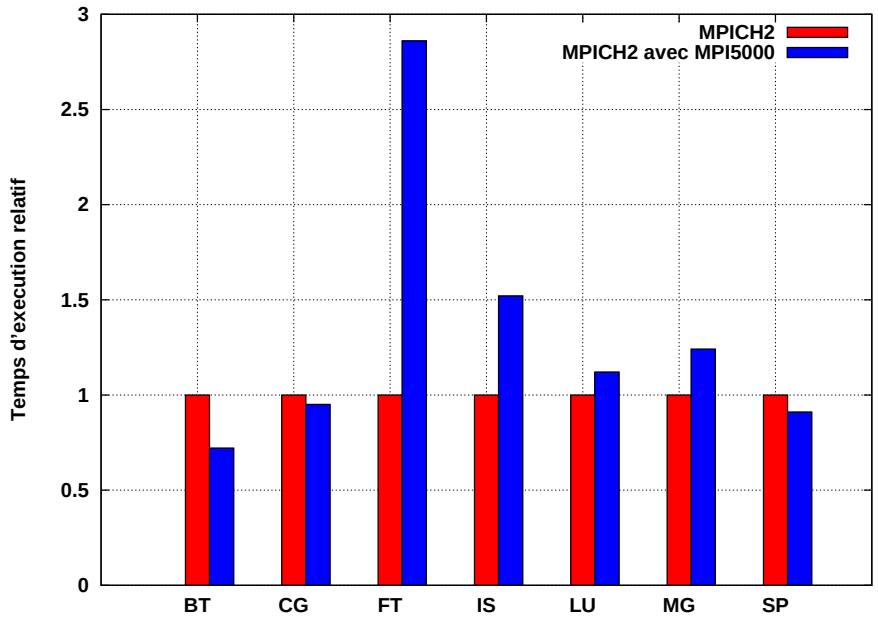


FIG. 5.8 – Temps d'exécution des NPB avec MPI5000 normalisés par MPICH2.

le cas de MPI5000, le terme "*local*" se réfère aux communications des noeuds vers les passerelles alors que le terme "*distant*" concerne la connexion longue distance entre les passerelles.

D'une manière générale, on constate que les signaux de congestion sont peu nombreux excepté pour FT et IS (qui communiquent beaucoup de manière synchrone). Comme nous l'avons vu dans la section 5.1.3.3, l'utilisation de passerelles doit contribuer à améliorer la détection des pertes sur le réseau longue distance. On remarque que le nombre de RTO diminue pour FT et IS comme prévu. Le nombre de DupAck diminue également sur le réseau longue distance pour IS mais augmente pour FT et pour les autres NPB avec MPI5000. Ceci est dû au goulot d'étranglement constitué par les passerelles. Aussi, ce n'est pas la réduction du nombre de retransmissions qui contribue aux améliorations de performances.

5.3.2.2 Diminution de nombre de démarrage lents après une période d'inactivité

Nous avons vu dans la section 4.2.3.2 que des applications comme BT et SP pouvaient tirer parti d'une désactivation du démarrage lent mais qu'il n'était pas envisageable de le désactiver pour toutes les applications car il engendrait des pertes des performances importantes pour les applications synchrones. Cependant, nous allons voir ici que l'éclatement des connexions mis en oeuvre par MPI5000 peut contribuer à réduire l'impact de celui-ci lorsqu'il n'est pas désactivé. Le tableau 5.7 montre le nombre de fois où la fenêtre de congestion décroit sans signal de congestion (sans détection de pertes) c.-à-d. le nombre de phases de démarrage lent après une période d'inactivité, avec et sans MPI5000 pour BT et SP. Les nombres pour MPICH2 avec MPI5000 ont été obtenus sur l'unique connexion longue distance entre les proxies. Les nombres pour MPICH2 sans MPI5000 sont une moyenne sur toutes les connexions longue distance. Ceci permet de s'abstraire des cas où toutes les connexions effectuent un démarrage lent au même

	MPICH2		MPICH2 avec MPI5000			
	Distant		Local		Distant	
NPB	DupAck	Timeouts (RTO)	DupAck	Timeouts (RTO)	DupAck	Timeouts (RTO)
BT	1	0	4	1	29	1
CG	1	0	2	0	10	0
FT	44	6	204	7	477	0
IS	275	20	21	1	53	0
LU	1	0	1	0	7	1
MG	0	0	0	0	0	0
SP	6	1	24	1	231	0

TAB. 5.6 – Nombre de signaux de congestion (détection de pertes) lors de l'exécution des NPB sans et avec MPI5000.

NAS	MPICH2 sans MPI5000	MPICH2 avec MPI5000
BT	331	323
SP	487	426

TAB. 5.7 – Nombre d'expirations du délai d'inactivité dans les NPB, sans et avec MPI5000

	MPICH2 sans MPI5000		MPICH2 avec MPI5000
NAS	Temps d'exécution (s)	Temps d'exécution sans activation du démarrage lent	Temps d'exécution (s)
BT	204	171	147
SP	242	203	221

TAB. 5.8 – Comparaison de l'exécution des NPB avec et sans démarrage lent après une période d'inactivité

moment. En effet, dans ce cas, le délai introduit sur chacune des connexions est simultané. Il ne faut donc pas additionner les délais mais ceci ne nous donne qu'une borne inférieure. Pour BT et SP, MPI5000 permet de diminuer le nombre de rédémarrage en phase lente après expiration du délai d'inactivité pour MPI5000.

Pour confirmer ces résultats, nous les comparons, dans le tableau 5.8, aux résultats de la section 4.2.3.2 lorsque nous avons désactivé le démarrage lent après une période d'inactivité. La désactivation améliore fortement les performances de BT et SP, dans des proportions comparables à celles de MPI5000. Ces résultats confirment que notre architecture contribue à réduire le nombre d'activations de l'algorithme. Cependant, le temps d'exécution inférieur de SP dans le cas avec désactivation montre que MPI5000 ne parvient pas à supprimer toutes les périodes de redémarrage lent.

5.3.2.3 Diminution du temps de détection des pertes

Les expériences précédentes n'expliquent pas l'impact de MPI5000 sur CG, LU et MG puisqu'ils ne présentent pas de période d'inactivité du point de vue de TCP. De plus, les valeurs obtenues dans le tableau 5.6 ne montrent pas de pertes (ni ACK dupliqués, ni RTO).

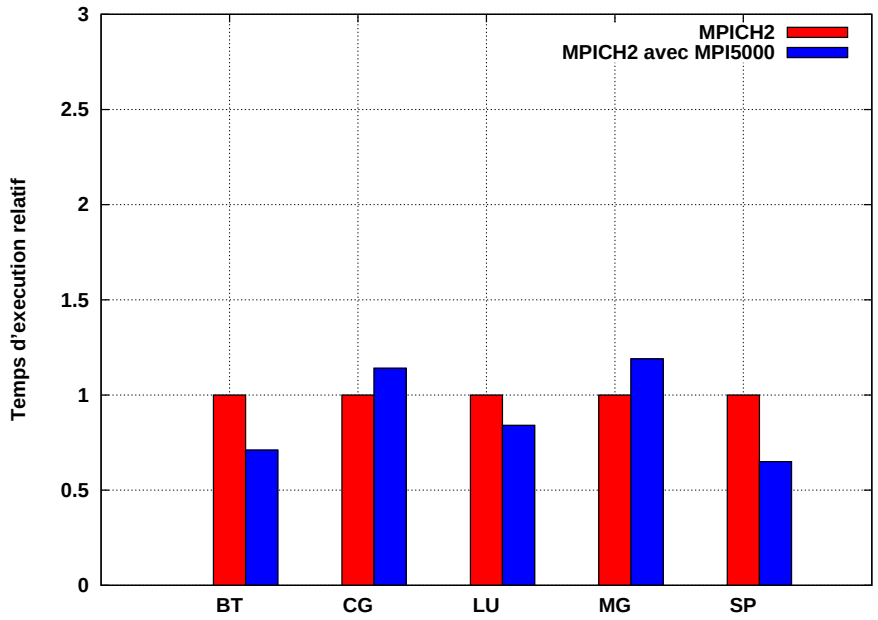


FIG. 5.9 – Temps d'exécution relatif des NPB avec trafic concurrent normalisé à MPICH2 avec trafic concurrent.

En effet, ces applications communiquent trop peu pour créer de la congestion. Mais pour que MPI5000 apporte une amélioration, il est nécessaire qu'il y ait des pertes (cf 5.1.3.3 et 5.1.3.4). Pour voir apparaître celles-ci, il est donc nécessaire d'ajouter du trafic concurrent, généré comme décrit précédemment dans la section 3.2.3.5 c.-à-d. avec deux flux `iperf` TCP en parallèle de l'application. Ces flux tentent d'utiliser la totalité de la bande passante disponible.

La figure 5.9 montre le temps d'exécution des NPB avec trafic concurrent et MPI5000 relatif au temps sans MPI5000 dans les mêmes conditions. On observe que BT, LU et SP bénéficient de notre architecture en cas de trafic concurrent. Cependant, CG et MG n'améliorent pas leur performances. Cela est probablement dû à leur type de trafic qui ne permet pas une augmentation assez rapide de la fenêtre de congestion sur la connexion longue distance mais de plus amples investigations sont nécessaires pour confirmer ce point.

Le tableau 5.9 montre, pour la même expérience, le nombre de détections de pertes (liés aux acquittements dupliqués ou au RTO) sans et avec MPI5000 dans le cas avec trafic concurrent. Les résultats avec BT, LU, MG et SP montrent une forte réduction du nombre de RTO et de DupACKs grâce à MPI5000. Cela confirme ce que nous avons supposé en section 5.1.3.3. Cependant, comme les communications de MG sont synchrones mais que la congestion qu'elles créent n'est pas assez importante, la réduction du nombre de DupACKs et de RTO ne parvient pas à dépasser le surcoût engendré par les recopies.

Ces résultats confirment nos hypothèses qui proposaient l'utilisation de passerelles pour réduire le nombre de retransmissions longue distance. Elles sont donc une approche valide pour améliorer les communications longue distance.

	MPICH2 sans MPI5000		MPICH2 avec MPI5000			
	Distant		Local		Distant	
NPB	DupAck	RTOs	DupAck	RTOs	DupAck	RTOs
BT	757	56	4	1	320	1
LU	327	232	0	0	174	41
MG	94	53	7	0	48	4
SP	1409	778	8	0	667	131

TAB. 5.9 – Nombre de DupACK et RTO sans et avec MPI5000 lors de l'exécution des NPB en présence de trafic concurrent.

5.3.2.4 Performances de MPI5000 avec une augmentation de la taille du problème

Pour confirmer les résultats des sections précédentes et adapter le problème aux capacités offertes par la grille, nous avons exécuté les NPB sur une taille plus conséquente (classe C) et sur un nombre plus important de noeuds. Notre banc d'essai était composé de 2 grappes de 32 noeuds reliées par un réseau à 1 Gbit/s. Les résultats pour FT et IS n'ont pas pu être obtenus.

La figure 5.10 montre les temps d'exécution des NPB sur 64 noeuds avec MPICH2 et MPI5000 relatifs aux temps d'exécution avec MPICH2 sans MPI5000. Aucun résultat ne montre d'amélioration avec MPI5000, MG est même très impacté. Cependant, nos passerelles ne sont pas optimisées et présentent donc un surcoût élevé. Il sera nécessaire d'effectuer des expérimentations supplémentaires pour comprendre où sont situés les coûts les plus importants et les réduire.

5.3.3 Changement de version de TCP sur la longue distance

Dans le chapitre 4, nous avions conclu que la nature du trafic nécessite d'utiliser l'une ou l'autre des variantes de TCP. Ainsi, si CUBIC est la plus appropriée pour les transferts de flux dans le cas où il y a congestion du lien, HTCP ou Illinois permettent d'obtenir les meilleures performances pour les applications MPI.

L'utilisation de MPI5000 contribue à modifier le trafic du réseau longue distance pour qu'il se rapproche d'un trafic constitué d'un flux continu. Il semble donc plus propice d'utiliser une variante appropriée sur le longue distance. De plus, il n'est pas possible d'utiliser simultanément plusieurs contrôles de congestion dans un même noyau linux. De ce fait, les communications locales et longue distance même si elles n'ont pas les même contraintes, utilisent la même variante de TCP. Les passerelles permettent de différencier le LAN et le WAN. Il est donc possible d'utiliser deux variantes de TCP différentes selon le type des communications (locales ou distantes). Les communications de la passerelle vers les noeuds utilisent obligatoirement la version longue distance. Ceci n'est pas un problème puisqu'il n'y a pas de congestion sur ce lien, celui ci étant de capacité supérieure ou égale au lien longue distance.

Dans le cas des passerelles, l'utilisation d'une variante moins aggressive (par exemple CUBIC) pour les communications LAN permettrait de diminuer les pertes qui se créent au niveau du goulot d'étranglement que constitue la passerelle (nombre de DupACKs et de TO). Sur le longue distance, une variante plus aggressive permettrait de rendre le trafic MPI plus compétitif par rapport aux autres flux qui s'exécutent en parallèle.

Nos expériences ont portés sur deux NPB différents : FT, pour ces communications synchrones avec des gros messages et SP, parce qu'il communique peu souvent mais beaucoup.

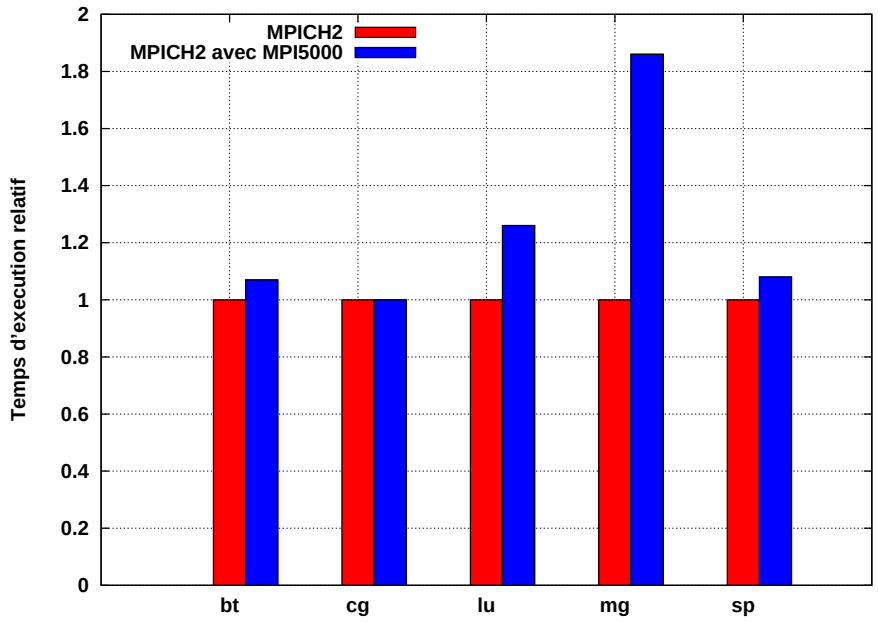


FIG. 5.10 – Temps d'exécution relatif des NPB (classe C) sur 64 noeuds normalisé à MPICH2 sans MPI5000.

	MPICH2		MPI5000		
NAS	BIC	CUBIC	BIC	CUBIC	CUBIC + BIC
FT	75	90	89	87	78
SP	131	135	125	119	117

TAB. 5.10 – Temps d'exécution des NPB avec et sans MPI5000, et en faisant varier les variantes de TCP.

Nous avons utilisé notre banc d'essai classique avec 8 noeuds sur chaque grappe, reliés par un lien à 1 Gbit/s avec une latence de 10.3 ms. Les passerelles utilisaient deux interfaces réseaux, l'une dédiée aux communications longue distance l'autre aux communications vers les noeuds. Nous avons utilisé deux variantes de TCP différentes : BIC et CUBIC.

Le tableau 5.10 montre les temps d'exécutions de FT et SP avec différentes variantes de TCP, avec ou sans MPI5000. On constate que l'utilisation d'une combinaison de variantes avec MPI5000 permet d'obtenir les meilleures performances, même si elle ne parvient pas à dépasser le temps sans l'utilisation de passerelles pour FT. Ces résultats montrent également qu'une variante appropriée pour le cas sans MPI5000 (BIC) est différente dans le cas avec MPI5000 (CUBIC). Ces expériences n'avaient pas pour but de trouver la meilleure combinaison de variante de TCP, mais de conclure sur la validité de l'approche. D'autres expérimentations seraient nécessaires pour trouver une combinaison optimale, notamment avec Illinois (qui n'était pas disponible au moment des tests).

Conclusion

Le chapitre 4 évaluait l'impact de TCP sur les communications MPI longue distance. Ce chapitre s'intéressait à réduire celui-ci. Nous nous sommes intéressés à un mécanisme d'éclatement des connexions TCP basé sur des passerelles introduites à l'interface LAN/WAN. Il permet de remplacer chaque connexion longue distance par trois connexions : deux connexions locales et une connexion longue distance. Ceci permet de pouvoir différencier les connexions pour les traiter différemment.

Dans un premier temps, nous avons montré comment l'éclatement des connexions permet de diminuer le temps de réponse de TCP. En effet, ce mécanisme diminue l'impact du contrôle de fiabilité sur les messages MPI en réduisant le temps de retour d'un acquittement puisque son émetteur est plus proche physiquement. Ce système permet également de garder la fenêtre de congestion de la connexion longue distance plus proche du débit réel du lien. De ce fait, le délai d'émission d'un message MPI est réduit, la fenêtre de congestion ne limitant pas inutilement le nombre de segments TCP émis en un RTT.

A la suite de cette étude, nous avons présenté une architecture à base de passerelles qui permet de réaliser l'éclatement des connexions TCP et donc des bénéficier des réductions des délais de transmission présentés précédement. Cette architecture, MPI5000, est basée sur une librairie et des passerelles, qui s'exécute sur tous les noeuds où l'application est executée entre MPI et TCP, et des programmes passerelles qui tourne sur un noeud spécifique. La librairie s'exécute sur tous les noeuds où l'application tourne, entre MPI et TCP. Les passerelles tournent sur un noeud spécifique dont le rôle est l'acheminement du trafic MPI longue distance. Ainsi, MPI5000 permet une adaptation transparente à toutes les implémentations MPI existantes, sans en modifier le code source.

Les résultats montrent que toutes les applications ne peuvent bénéficier de ce mécanisme car le gain généré par l'accélération des retransmissions ne parvient pas toujours à compenser le surcoût provoqué par les passerelles. Ainsi, les applications aux communications synchrones sont particulièrement impactées par ce système. Cependant, dans nos tests, les passerelles ne disposaient que d'une seule interface, ce qui peut conduire à diviser par deux la bande passante disponible. D'autres applications parviennent à des améliorations de performances, certaines très conséquentes comme SP dans le cas de trafic concurrent (35%). Enfin, l'évalaution de notre architecture sur des applications plus importante en termes de calculs et de communication, sur deux sites différents et 64 noeuds, qu'il est nécessire de réduire les temps de retransmission. Notre approche est donc valide mais nécessite d'avoir des passerelles optimisées capables d'émettre au débit du lien.

Chapitre 6

Conclusion

Les grappes sont des architectures pour l'exécution d'applications parallèles qui relient plusieurs ordinateurs par un réseau de manière à les faire travailler ensemble. Lorsque ces applications ont eu besoin de ressources supplémentaires, il a été nécessaire de regrouper les grappes, situées sur des sites séparés, en les interconnectant par un réseau longue distance. Ainsi, sont apparues les grilles.

La plupart des applications parallèles sont écrites à l'aide du standard MPI (Message Passing Interface) qui fonctionne par passage de messages entre processus. Alors que les communications intra-sites des applications MPI peuvent disposer d'un protocole dédié pour le réseau d'interconnexion des grappes, les communications sur le réseau longue distance utilisent principalement TCP comme protocole de transport.

La problématique générale de nos travaux était donc d'exécuter au mieux des applications MPI sur une grille de calcul avec TCP comme protocole de transport sur le réseau longue distance, en optimisant l'interaction entre ces deux couches de communication.

6.1 Contributions

Le réseau longue distance des grilles de calcul impose de nouvelles contraintes : gestion de la latence élevée, gestion du goulot d'étranglement qu'il peut constituer pour les applications qui communiquent dessus et gestion du partage entre les différents utilisateurs. Le chapitre 1 identifiait les problématiques que nous avons traitées par la suite. Afin d'identifier les points problématiques des communications et de pouvoir optimiser l'interaction entre MPI et TCP, il nous a été nécessaire de comprendre comment les applications MPI communiquent sur le réseau longue distance et comment les contraintes de ce réseau influent sur ces applications. Ensuite, nous avons cherché à identifier comment les mécanismes de TCP posent problème à l'exécution des applications MPI sur la grille. Enfin, nous nous sommes intéressés à réduire l'impact de TCP sur les communications longue distance de telles applications.

Le chapitre 2 présentait les nombreuses implémentations MPI ont été proposées pour l'exécution d'applications dans la grille. Cependant, peu d'entre elles explorent et optimisent les communications sur le réseau longue distance. GridMPI est la plus avancée dans ce domaine mais n'étudie pas en détail comment le contrôle de congestion ou le contrôle de fiabilité de TCP peuvent influer sur les communications MPI.

Dans le chapitre 3, nous avons présenté deux outils, une librairie (`InstrAppli`) et un module noyau (`tcpprobe` modifié), permettant d'obtenir conjointement les paramètres qui pouvaient retarder l'envoi de messages MPI, respectivement au niveau de l'API socket et de la couche TCP. `InstrAppli` nous permet d'obtenir une analyse détaillée des communications des applications MPI : leur schéma de communications, l'évolution temporelle des écritures sur la socket et une vue aggrégée des caractéristiques des communications longue distance. Le module `tcpprobe`, quand à lui, nous donne l'évolution de la fenêtre de congestion et du tampon de réception de TCP au cours du temps. L'utilisation simultanée des deux outils nous fournit une vue globale assez complète des communications d'une application MPI.

Nous avons utilisé ces outils pour obtenir les caractéristiques des NPB (Nas Parallel Benchmark), des programmes représentatifs de la variété des applications MPI. Nous en avons dégagé une classification, à plusieurs niveaux, en fonction de leurs communications longue distance. Nous avons différencié les applications synchrones ou non (tous les processus communiquent en même temps), celles qui communiquent beaucoup ou pas beaucoup (taille des messages), et celles qui communiquent souvent ou peu souvent (temps entre deux messages). A l'aide de ces informations, nous avons montré que la grille, en apportant des ressources supplémentaires

pouvait être un environnement valide pour l'exécution d'applications MPI, puisque celles-ci parviennent à améliorer leur temps de complétion. Une étude plus précise nous a permis d'évaluer l'impact de chacun des points problématiques du réseau longue distance de la grille : la latence ralentit principalement les applications qui communiquent souvent alors que la congestion — créée soit par l'augmentation du goulot d'étranglement, soit du fait du trafic concurrent —, affecte principalement les applications aux communications synchrones.

Comme l'augmentation du temps d'exécution sur la grille n'était pas dû uniquement aux caractéristiques physiques de la grille mais aussi aux caractéristiques des communications des applications, nous nous sommes intéressés aux mécanismes du protocole de transport TCP, qui pouvaient là aussi limiter les applications MPI.

Dans le chapitre 4, nous avons étudié deux mécanismes de TCP qui pouvaient limiter les performances de telles applications : le contrôle de congestion et le contrôle de fiabilité.

Le contrôle de congestion est réalisé par une fenêtre de congestion qui limite le nombre de données en vol sur une connexion pour éviter la congestion. Si celle-ci n'est pas adaptée aux conditions réelles du réseau, elle peut ralentir l'émission des messages MPI. Cependant, nous avons montré que le contrôle de congestion ne pouvait être supprimé dans une grille de calcul même si les communications sont peu nombreuses. Son absence engendre trop de pertes au niveau des commutateurs des noeuds récepteurs dont le coût n'arrive pas à surpasser le gain obtenu par la suppression de la fenêtre de congestion. Nous avons montré également que le démarrage lent du contrôle de congestion affecte particulièrement les communications MPI qui sont courtes. Le démarrage lent est utilisé au début d'une connexion ou après un temps d'inactivité. Le délai d'inactivité affecte uniquement les applications MPI qui communiquent peu souvent. La désactivation du redémarrage lent après une période d'inactivité permet d'obtenir une amélioration des temps d'exécution des applications qui communiquent peu souvent et de manière non synchrone. Les applications synchrones subissent trop de pertes, et ce mécanisme est justifié. Toutefois, une désactivation de ce mécanisme entraine une augmentation conséquente des temps de transfert de fichiers qui s'exécutent en parallèle des applications MPI. En effet, après une période d'inactivité, les applications recommencent à émettre avec la dernière taille de fenêtre de congestion connue. Celle-ci n'est pas forcément adaptée au debit actuellement disponible sur le réseau et va émettre à un débit supérieur, contribuant à créer des pertes pour le trafic concurrent.

Différentes variantes de TCP ont été proposées pour faire évoluer la fenêtre de congestion dans le cadre de flux continus tels que les transferts de fichiers. Celles-ci proposent d'améliorer l'utilisation de la bande passante disponible sur un lien avec un grand produit débit * délai. Nous avons montré que deux de ces variantes étaient particulièrement adaptées pour les applications MPI : Illinois et HTCP. En effet, elles permettent à la fenêtre de congestion de retrouver rapidement une valeur proche de sa taille avant une perte, tout en veillant à ne pas la dépasser ce qui engendrerait de nouvelles pertes.

L'étude du contrôle de fiabilité de TCP nous a permis de montrer que les pertes sont plus coûteuses pour les applications MPI que pour les transferts de fichiers. En effet, si une perte intervient dans la dernière partie d'une communication — qui correspond aux données émises dans une période correspondant au temps final sans perte moins un RTT —, le recouvrement de celle-ci est particulièrement lent. Aussi, comme la plupart des communications MPI sont effectuées en un temps inférieur au RTT, une perte affecte plus ces applications que des transferts de fichiers. De ce fait, une évolution de la fenêtre de congestion qui minimise les pertes durant cette période particulière (en fin de communication) est préférable. Ces résultats confirment une partie de nos résultats précédents, puisque dans la variante Illinois les pertes les plus probables ont lieu au début des communications. En effet, après une perte, cette variante

augmente la fenêtre de congestion de manière rapide au début puis de plus en plus lentement. Elle est donc mieux adaptée aux communications MPI.

Les réseaux locaux et longue distance ont des caractéristiques différentes, notamment en terme de latence et de bande passante, ce qui a un impact direct sur les mécanismes de contrôles de TCP. Nous avons donc cherché à différencier les deux types de réseaux, de manière à optimiser spécifiquement les communications sur le longue distance. Dans le chapitre 5, nous avons proposé un mécanisme d'éclatement des connexions TCP basé sur des passerelles à l'interface LAN/WAN. Il permet de séparer chaque connexion longue distance par trois connexions : une connexion LAN/LAN, une connexion WAN-WAN et une connexion LAN/LAN. Ainsi, nous pouvons traiter spécifiquement les connexions WAN-WAN.

Nous avons montré que ce mécanisme pouvait améliorer le réactivité du contrôle de congestion. Les passerelles permettent de détecter et retransmettre les données perdues. En effet, ce sont les passerelles qui détectent la perte après un RTT local au lieu du noeud distant après un RTT longue distance. L'utilisation de passerelles permet d'aggréger l'ensemble du trafic du réseau longue distance sur une unique connexion. Ceci permet d'évaluer plus régulièrement le débit disponible sur ce lien puisque toutes les données qui transitent entre les sites contribuent à mettre à jour la fenêtre de congestion. Ainsi, le nombre de démarrages lents est diminué et la fenêtre de congestion est plus proche du débit réellement disponible.

Basée sur ce principe, nous avons proposé MPI5000, une architecture comprenant une librairie et des passerelles, qui permet de réaliser l'éclatement des connexions TCP. MPI5000 s'adapte de manière transparente à toute implémentation MPI. La librairie permet de rediriger les connexions TCP des implémentations MPI vers les passerelles, en interceptant les appels des fonctions de la librairie standard. Les passerelles retransmettent les données MPI du réseau local vers le site distant, et vice versa.

L'évaluation de notre architecture montre la validité de notre approche : le nombre de pertes sur le réseau longue distance est diminué, le nombre de redémarrage lent également. Cependant, les performances ne sont pas toujours améliorées particulièrement dès que les applications communiquent de manière synchrone. Dans ce cas, les recopies introduites sur les passerelles deviennent coûteuses. Une optimisation des passerelles est donc nécessaire pour parvenir à ce que leur apport soit supérieur au surcôut qu'elles induisent. L'éclatement des connexions TCP va permettre de changer la nature du trafic MPI du réseau longue distance pour le rendre plus proche de flux continus. Ceci, combiné à la suppression de la contrainte technique, permet de pouvoir utiliser deux variantes de TCP différentes selon le réseau (WAN ou LAN). Nous avons montré que cette approche permet d'obtenir de meilleurs temps d'exécution que l'utilisation de la même variante sur les deux réseaux.

6.2 Perspectives

Les travaux présentés dans cette thèse ouvrent de nombreuses perspectives à court, moyen ou long termes pour une exécution encore plus efficace des applications MPI sur des grilles de calcul.

Optimisation des programmes passerelles

Nous avons vu que les programmes passerelles étaient le principal facteur limitant de l'éclatement des connexions TCP : les recopies qu'ils induisent peuvent avoir un coût rédhibitoire pour l'application. Au niveau de la passerelle, seule l'entête MPI5000 contient des données utiles permettant d'effectuer le routage des messages. Nous avons donc envisagé de ne copier que l'entête en espace utilisateur, laissant les données dans le tampon de réception de TCP. Un

appel à une fonction du noyau permet de copier les données d'un tampon de TCP (réception) à l'autre (émission). Cependant, nos efforts dans ce domaine n'ont pas aboutit, la recopie en espace utilisateur étant toujours plus rapide. Nous allons poursuivre nos efforts dans cette direction. Il est également envisageable de réaliser la retransmission des messages au niveau TCP : ainsi les messages seraient copiés directement d'un tampon TCP à l'autre, voir éviter cette recopie en utilisant un seul tampon commun.

Pour permettre d'améliorer les performances, il est également possible d'utiliser plusieurs passerelles pour limiter leur surcoût lorsqu'une grande quantité de données à retransmettre arrive simultanément. En effet, les recopies s'exécuteraient en parallèle, et l'impact des passerelles serait donc limité aux données transmises par une seule d'entre elles. Ainsi, nous pourrions proposer notre architecture pour toutes les applications (MPI ou autres) dont les communications sur des réseaux aux fortes latences sont courtes.

Limitation de débit sur le réseau longue distance

La limitation de débit sur le réseau longue distance permettrait de s'abstraire du contrôle de congestion tout en garantissant un débit identique à toutes les applications. Cependant, il est difficile de limiter le débit des applications MPI sans les pénaliser du fait de leur mode de communication. Comme leur débit n'est pas constant, il est, en effet, difficile de déterminer le débit optimal d'émission. Nous avons vu que les passerelles, du fait qu'elles aggrègent les connexions longue distance, permettent de se rapprocher d'un flux continu. Aussi, la limitation de débit aurait moins d'impact sur les applications MPI.

L'utilisation d'une passerelle au niveau de chaque site pour toutes les applications de grille permettrait également de réguler les pertes. Ainsi, la totalité des connexions longue distance utiliserait la passerelle. De ce fait, le partage de la bande passante du lien longue distance, serait régulé par les passerelles. Tous les trafics obtiendraient donc une bande passante similaire, tout en laissant la totalité de la bande passante aux autres communications lorsqu'il n'y a pas de trafic MPI.

Intégrer le placement des tâches dans MPI5000 à l'aide d'InstrAppli

Nous avons vu que InstrAppli nous permettait d'obtenir de nombreuses informations sur les communications longue distance. Une première exécution de l'application permet de récolter ces informations. Aussi, on pourrait envisager d'effectuer un placement des processus MPI sur les noeuds en fonction des communications, lors du lancement du `mpirun` avec MPI5000. Le placement de tâches sur une grille à déjà été grandement étudié, mais notre architecture offre une possibilité simple de le mettre en oeuvre.

Modélisation des communications MPI sur TCP

L'impact du contrôle de congestion est à la fois positif et négatif pour les communications MPI : il ralentit l'émission d'un message mais contribue à éviter les pertes qui elles aussi ralentissent les applications. Il est donc nécessaire de trouver le juste débit, qui est idéalement situé entre les deux.

L'étude théorique que nous avons présenté dans le chapitre 4 utilisait le délai introduit par une perte pour évaluer l'impact du contrôle de fiabilité sur les applications MPI. Aussi, il serait nécessaire d'ajouter à cette étude le délai introduit par une fenêtre de congestion qui impose d'envoyer un message en plusieurs fois et la probabilité de perte si une fenêtre de congestion est trop grande. Ainsi, on obtiendrait un modèle précis de l'impact de TCP sur les communications MPI sur un réseau de grille. Ceci nous permettrait de définir la variante de TCP la plus adaptée à notre problématique ou d'en proposer une nouvelle, si aucune ne convient.

Adapter le protocole de transport

Nous avons vu que le temps de réponse du protocole de transport était très couteux pour

les applications avec des communications dont le temps d'émission est inférieur au RTT. Pour l'améliorer, il faut soit détecter les pertes plus rapidement, soit les éviter au maximum pour les communications courtes.

Le premier point implique de réduire la distance entre la perte et l'équipement qui la détecte. Notre approche à base de passerelles est une possibilité mais il est également possible de s'appuyer sur les équipements réseau pour obtenir cette information au plus tôt. Le protocole XCP [KHR02] est basé sur cette approche mais sa mise en oeuvre est difficile parce qu'elle nécessite que tous les équipements coopérent. Le second point demande d'avoir connaissance, au préalable, de la durée d'une communication. On pourrait envisager de gérer des priorités pour les communications selon leur fréquence et leur taille. Plus l'espace entre deux émissions est grand plus la probabilité d'une perte serait donc faible. Ainsi, une connexion qui émet de petits messages peu fréquents subirait moins de pertes qu'un flux continu. Ce mécanisme pourrait être ajouté à des mécanismes de gestion de la qualité de service déjà disponibles.

La plupart de ces optimisations nécessitent de s'appuyer sur les équipements réseaux sous-jacents (couche réseau) afin d'accélérer la mise à jour de l'information de l'état du réseau. Il est également nécessaire de collecter des informations sur les connexions et les données que l'on transfere dessus (au niveau de la couche TCP). Aussi, il semble nécessaire de proposer des mécanismes permettant des échanges entre la couche transport et la couche réseau du modèle OSI pour améliorer la collaboration entre les deux couches.

Appendices

Annexe A

Fonctionnement de TCP

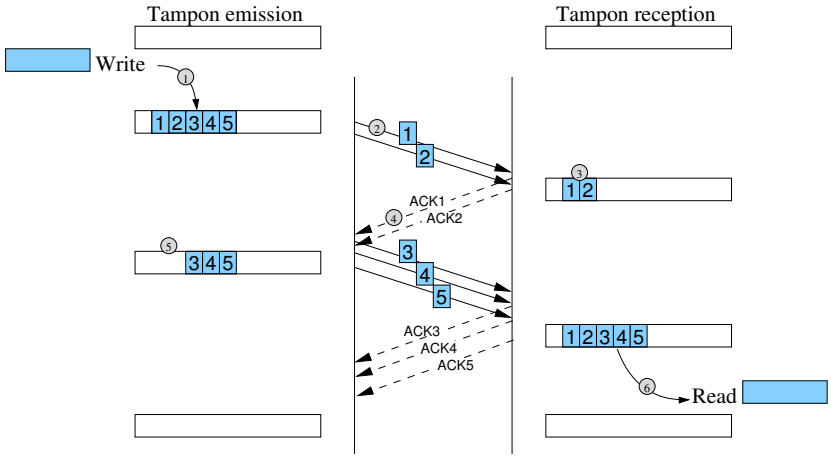


Fig. A.1 – Fonctionnement de TCP en état stable

A.1 Principe général de TCP

TCP [Ins81] (Transmission Control Protocol), conçu il y a 20 ans pour Internet, est un protocole de transport fiable de bout en bout, qui a pour but de partager équitablement, entre plusieurs flux, la bande passante disponible sur un lien. Il fonctionne en mode connecté c'est à dire que les deux parties d'une connexion, — l'émetteur et le récepteur —, établissent, contrôlent et ferment cette connexion. Une connexion est identifiée par deux couples *(adresse IP, port)*, un pour la source, un pour la destination. Ce protocole est fiable puisque qu'il garantit à l'application que les données qu'elle transmet arriveront à destination, sans erreur et sans pertes. L'unité de transfert est le segment TCP, qui contient notamment les ports source et destination, un numéro d'ordre qui identifie le segment envoyé, une somme de contrôle ainsi que les données envoyées. Sur un réseau Ethernet classique, la taille des segments TCP permet de transporter 1448 octets de données utiles. TCP utilise deux tampons mémoire de chaque coté de la connexion, —un pour l'émission, un pour la réception— pour stocker les segments en attente d'émission et en attente de réception (seuls les tampons utilisés sont représentés sur la figure A.1). La figure A.1 montre le cheminement des paquets TCP. Les données sont d'abord stockées dans le tampons d'émission et découpées en segments (1) puis émises vers le

destinataire (2). Lorsqu'un paquet est reçu par le récepteur, celui-ci va stocker le paquet dans le tampon de réception (3), en attendant que l'application destinataire recopie les paquets (6). Dans le même temps, il émet un accusé de réception (4) (sous forme d'un paquet spécifique appelé ACK) pour signifier à l'émetteur le numéro de séquence du dernier paquet reçu dans l'ordre. Finalement, lorsque l'accusé arrive du coté de l'émetteur, ce dernier supprime le paquet du tampon d'émission (5).

A.2 Contrôle de fiabilité

La détection d'une perte peut se faire de deux manière différentes comme le montre la figure A.2.

- Cas 1 : Lorsque le récepteur reçoit trois accusés identiques, il suppose qu'il y a eu une perte et ré-émet le paquet en activant l'algorithme de FastRetransmit (retransmission rapide) décrit dans la RFC2581 [APS99]. Sur la figure, l'émetteur S_1 envoi les paquets 1, 2, 3 et 4. Lors de la réception des paquets 1, 3 et 4, TCP émet un accusé correspondant au paquet 1. Lorsque le récepteur reçoit les trois accusés correspondant au paquet 1, il considère que les paquets émis après le paquet 1 ont été perdus et les retransmet. S'il y a suffisamment de paquets émis après le paquet perdu, la retransmission intervient après un RTT (Round Trip Time), c'est à dire entre 5 ms et 100 ms dans les grilles actuelles.
- Cas 2 : Si aucun paquet n'est émis après le paquet perdu (le paquet 12 sur la figure), il est nécessaire d'attendre l'expiration du délai de retransmission (nommé RTO, Retransmission Timeout). Ce délai est fonction du RTT mais lui est très supérieur pour éviter les retransmissions inutiles. Dans l'implémentation actuelle de TCP sous Linux, le RTO vaut 200 ms plus la valeur du RTT.

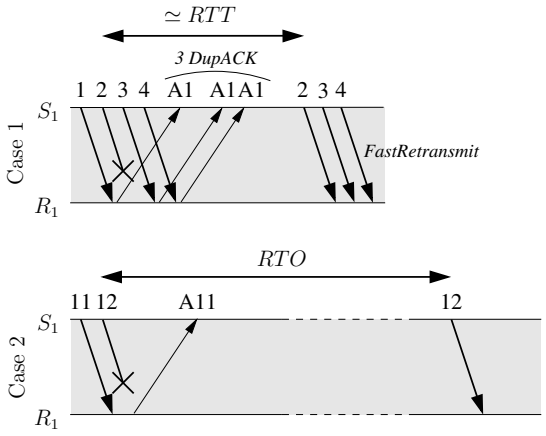


FIG. A.2 – Comportement de TCP en cas de perte

A.3 Contrôle de fiabilité

Le contrôle de congestion est décrit dans la section 4.2.1.

Annexe B

Détails d'implémentation des outils de tracage

Cette section détaille les caractéristiques des outils de traçage des communications des applications MPI. Ces informations peuvent être relevées au niveau des différentes couches traversées lors de l'envoi de messages MPI : au niveau de l'implémentation MPI ou au niveau du noyau (principalement de la couche TCP).

B.1 Au niveau de l'implémentation MPI

B.1.1 TAU

TAU [SM06] est un outil de récupération de trace d'exécution d'applications parmi Fortran, C, C++, Java, Python. L'instrumentation est réalisée à l'aide du script *tau_ instrumentor* ou à la compilation avec un compilateur intégré au logiciel. Cependant, pour les applications MPI, TAU peut être chargé au lancement de l'application à l'aide de LD_PRELOAD si la librairie MPI est compilée dynamiquement.

TAU peut générer des traces au format SLOG2 ou Vampir. La visualisation est réalisée avec ParaProf si on veut le profil d'exécution ou Jumpshot pour une visualisation de la trace.

B.1.2 Pajé

Pajé [CdKdOS00] est un outil de visualisation de trace générée par une version instrumentée de la librairie ATHAPASCAN. La synchronisation est réalisée par une horloge globale précise. La visualisation inclut à la fois les nœuds et les threads exécutés sur chaque nœuds. Elle fournit également des statistiques sur les activités des nœuds. Les communications sont classiquement représentées par des flèches.

Un autre projet utilise la librairie MPIRastro de manière à instrumenter le code MPI de manière transparente en liant l'application avec la librairie lors de l'édition des liens. La trace obtenue est ensuite visualisée avec Pajé de la même manière que précédemment. Une évolution de Pajé est également proposée par Schnorr [SHON08] de manière à visualiser en 3D l'application. Les axes x et y fournissent une représentation spatiale des processus sur la grille, alors que l'axe z représente le temps. Chaque communication est représentée par une ligne entre deux processus. Le regroupement des processus par site est effectué grâce à la topologie de la grille qui est fournie à l'application.

B.1.3 SvPablo

SvPablo [RZR98] est un autre outil pour l'instrumentation de code source et la visualisation de la trace. L'instrumentation est réalisée au choix de manière interactive avec l'utilisateur ou automatiquement (pour les programmes Fortran uniquement). La visualisation des performances est effectuée en regard du code source, avec un indicateur du temps passé dans chaque fonction.

B.2 Au niveau noyau

B.2.1 KTAU

KTAU [NMSM06] propose une instrumentation du noyau Linux afin d'extraire les appels à une grande partie de ses fonctions —dont certaines fonctions de communications mais pas uniquement. L'utilisation de KTAU est décidée à la compilation du noyau, par activation de deux paramètres dans le noyau. Le temps passé dans chaque fonction est stocké dans */proc/ktau* et un démon se charge d'extraire les valeurs périodiquement. TAU est implémenté comme étant une librairie de KTAU de manière à permettre l'utilisation conjointe des deux outils. Les deux outils utilisent une horloge matérielle précise de manière à synchroniser les événements au niveau utilisateur comme au niveau noyau. KTAU utilise également Jumpshot pour visualiser les traces sous forme de diagramme de Gantt.

B.2.2 Web100

Web100 [MHR03] a été développé dans le but de fournir aux ingénieurs qui développent des infrastructures une vision de ce qui ce passe au niveau de la couche réseau, de manière à améliorer les performances des couches supérieures. Web100 se compose à la fois d'un patch noyau permettant d'instrumenter le code de la pile TCP et d'un ensemble d'outils permettant de visualiser les variables exposées. L'ensemble de ces instruments a été décrit dans la RFC 4898. Ainsi, Web100 stocke les événements de toutes les connexions TCP telles que la taille de la fenêtre de congestion, le nombre de pertes, le nombre de retransmissions, etc.

B.2.3 Le module tcpprobe

Ce module enregistre l'état d'une connexion TCP lors de l'arrivée de paquets (données ou ACK). Il fonctionne par l'insertion d'un détournement (hook) dans la pile de réception de TCP à l'aide de *kprobe* de manière à capturer la fenêtre de congestion et les numéros de séquence des paquets TCP.

B.3 Synthèse

Au niveau MPI, l'outil le plus approprié est TAU, mais il souffre de l'absence d'une vision globale des communications effectuées. De la même manière, Pajé ne propose pas d'agrégation des communications. Sa version 3D propose cependant une idée intéressante de visualisation des communications par une représentation spatiale.

Du coté des communications TCP, Web100 fournit l'ensemble des variables intéressantes. Cependant, il ne fournit pas l'évolution de ces valeurs au cours du temps. Il est donc nécessaire de relever ces valeurs périodiquement mais ceci peut impacter les performances du système. D'un autre coté, tcpprobe consigne effectivement les événements au cours du temps mais il

n'est pas suffisant en l'état vu qu'il ne nous fournit que l'évolution de la fenêtre de congestion, et les numéros de séquence des paquets TCP. KTAU quant à lui ne se préoccupe que des appels aux fonctions du noyau et ne permet de visualiser la fenêtre de congestion au cours du temps ou le nombre de retransmissions. Par contre, l'utilisation conjointe de KTAU et TAU permet d'avoir une vision beaucoup plus générale des communications au niveau des deux couches qui nous intéressent.

L'idéal serait donc un outil, qui à la manière de TAU et KTAU, intègre des informations sur les deux couches de communications. Du coté MPI, il faut pouvoir agréger les informations par paire de processus (en quelque sorte, réaliser du profiling sur une connexion) et pouvoir choisir les connexions qui nous intéressent. Du coté TCP, il est également nécessaire d'obtenir des informations sur les événements qui limitent les émissions tels que les phases de slowstart, l'évolution de la fenêtre de congestion et le nombre de retransmission.

Du point de vue des outils, il n'existe pas d'outil approprié permettant de différencier les communications par type c'est à dire d'aggréger les information selon le réseau qu'elles traversent (local ou longue distance). De plus, les informations dont on dispose au niveau TCP, ne permettent pas une étude fine des paramètres qui vont influer les performances.

Annexe C

Algorithmes de MPI5000

Algorithm 1: Algorithme des programmes passerelles

```
Input: s : site/gateway number, P_s, port_d_s : listening for other gateways, port_l_s :
       listening port for nodes
/* listening local connections                                          */
bind(port_l_s);
if erreur then redo and modifing E2 fdl = listen(port_l_s);
/* connecting gateways together                                         */
bind(port_d_s);
if s ≠ 1 then
    fdw = listen(port_d_s);
end
for j decreasing from S to s+1 do
    tant que pas fait faire
        fdw_{j,s} = accept(fdw);
    fin
end
for i from 1 to s-1 do
    while not done do
        fdw_{s,i}= connect(i, port_d_i)
    end
end
/* accepting local connections                                          */
nb_conn_lan = 0;
while nb_conn_lan < P_s do
    ftemp = accept(fdl, addr);
    s,n,p = findId(addr, E1);
    fdl_{s,n,p} = ftemp;
    port_{s,n,p} = read(fdl_{s,n,p}, 4);
    nb_conn_lan++;
end
/* Waiting and retransmission                                           */
while select(fdw_*, s, fdl_s, *, *) do
    if something on fd then
        /* header = flag, dest, len, src. Cf. 5.2.3                     */
        header = read(fd, header_size);
        while !all data read do
            data = read(fd, max(MAX_CHUNK, len));
            if x = s then
                /* transmission to local nodes                          */
                if connect then
                    connect(connect_fd_{x,y,z}, findIP(x,y), findPort(x,y,z));
                    port_{x,y} = read(fd_{x,y,z})
                else
                    write(fdl_{x,y,z}, len + src + data);
                end
            else
                /* transmission to another gateway                      */
                write(fdw_{s,x}, header + data);
            end
        end
    end
end
```

Algorithm 2: Algorithme de la librairie

```
/* Function ConnectToGw()                                              */
begin
    if no fd to the gateway then
        /* p is incremented for each process                           */
        s, n, p = findId(E1)
        port_l_s = findPort(s, E2)
        fdl_{s,n,p} = connect(s, port_l_s)
        port_{s,n,p}=getPort(fdl_{s,n,p})
        /* Inform gateway of the port ID                               */
        write(fdl_{s,n,p}, port_{s,n,p})
    end
end

/* Interception of bind, connect, accept, poll, read, write           */

/* Bind(in : fd_ecoute, in : port_{s,n,p})                             */
ConnectToGw()

/* Connect(in : ip_{x,y}, in : port_{x,y,z})                           */
if connect distant then
    ConnectToGw()
    fdLibre = rechercheFd()
    listeFd[fdLibre] = x, y, z
    len = 4 + 4 + 4; /* 1 indicate a propagated connect                */
    write(fdl_{s,n,p}, x, y, z + 1 + len + s, n, p + ip_{s,n} + port_{s,n,p})
    return fdLibre
else
    /* local connection                                                */
    return connect()
end

/* Accept(in : fd_ecoute, out : addr_cli)                              */
/* accept is realised even for a distant connect but this socket will
    never be used                                                       */
if accept distant then
    fd = accept()
    dest = read(fd)
    write(fd, fd_port)
    return fd
else
    return accept()
end
```

Algorithm 2 : Nodes algorithm (following)

```
/* Read(in : fd)                                                    */
if fd in listeFd then
    /* distant connection : reading on fdl_{s,n,p}                  */
    if full[fd] then full[fd] = false;
    return data[fd];
    x',y',z' = findId(fd);
    repeat
        len = read(4);
        x,y,z + data[fdl_{x,y,z}) = read(fdl_{s,n,p}, len)
    until x',y',z' ≠ x,y,z ;
    full[fd] = false;
    return data[fdl_{x,y,z}]
else
    return read(fd)
end

/* Write(in : fd)                                                   */
if fd dans la table then
    /* distant connection : writing on fdl_{s,n,p}                  */
    /* See subsection 5.2.3 for protocol description                */
    x,y,z=findId(fd);
    return write(fdl_{s,n,p}, x,y,z + (len + 4) + s,n,p + data)
else
    return write(fd)
end

/* Close(in : fd)                                                   */
if fd in listeFd then
    /* distant connection                                           */
    listeFd[fd]=0;
    nbFd–;
    if nbFd = 0 then close(fdl_{s,n,p})
else
    close(fd);
end
```

Références

Articles

[ABFN07] Olivier Aumage, Elisabeth Brunet, Nathalie Furmento, and Raymond Namyst. Newmadeleine : a fast communication scheduling engine for high performance networks. In *CAC 2007 : Workshop on Communication Architecture for Clusters, held in conjunction with IPDPS 2007*, Long Beach, California, USA, March 2007.

[ABTV06] Eitan Altman, Dhiman Barman, Bruno Tuffin, and Milan Vojnovic. Parallel TCP Sockets : Simple Model, Throughput and Validation. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–12, April 2006.

[AM03] Olivier Aumage and Guillaume Mercier. MPICH/MADIII : a Cluster of Clusters Enabled MPI Implementation. In *Proceedings of CCGrid*, Tokyo, 2003.

[Aum02] Olivier Aumage. Heterogeneous multi-cluster networking with the Madeleine III communication library. In *16th International Parallel and Distributed Processing Symposium*, Florida, USA, April 2002.

[BBB+94] D Bailey, E Barscz, J Barton, D Browning, R Carter, L Dagum, R Fatoohi, S Fineberg, P Frederickson, T Lasinski, R Schreiber, H Simon, V Venkatakrishnan, and S Weeratunga. The NAS Parallel Benchmarks. Technical Report RNR-94-007, NASA Ames Research Center, Moffett Field, California, USA, 1994.

[BCC+06] Raphaël Bolze, Franck Cappello, Eddy Caron, Michel Daydé, Frédéric Desprez, Emmanuel Jeannot, Yvon Jégou, Stephane Lantéri, Julien Leduc, Noredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, Benjamin Quetier, Olivier Richard, El-Ghazali Talbi, and Touche Iréa. Grid'5000 : a large scale and highly reconfigurable experimental Grid testbed. *International Journal of High Performance Computing Applications*, 20(4) :481–494, November 2006. https://www.grid5000.fr/.

[BDV94] Greg Burns, Raja Daoud, and James Vaigl. LAM : An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium*, pages 379–386, 1994.

[BGP+94] Mike Barnett, Satya Gupta, David G. Payne, Lance Shuler, Robert van de Geijn, and Jerrell Watts. Building a high-performance collective communication library. In *Supercomputing '94 : Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, pages 107–116, New York, NY, USA, 1994. ACM.

[BH09] Carlos Jaime BARRIOS HERNANDEZA. *A study of grid architecture behavior for large data transfers*. PhD thesis, Nice Sophia Antipolis University, July 2009.

[BLM04] David P. Bunde, Vitus J. Leung, and Jens Mache. Communication patterns and allocation strategies. *Parallel and Distributed Processing Symposium, International*, 15 :248b, 2004.

[C+07] Charlie Catlett et al. TeraGrid : Analysis of Organization, System Architecture, and Middleware Enabling New Types of Applications. In Ed. Lucio Grandinetti, editor, *HPC and Grids in Action*. IOS Press 'Advances in Parallel Computing' series, 2007.

[CC06] D. Lacamera C. Caini, R. Firrincieli. PEPsal : a Performance Enhancing Proxy designed for TCP satellite connections. In *IEEE 63rd Vehicular Technology Conference (VTCSpring06)*, May 2006.

[CdKdOS00] Jacques Chassin de Kergommeaux and Benhur de Oliveira Stein. Pajé : An Extensible Environment for Visualizing Multi-threaded Programs Executions. In *Euro-Par '00 : Proceedings from the 6th International Euro-Par Conference on Parallel Processing*, pages 133–140, London, UK, 2000. Springer-Verlag.

[CDS00] Bradford L. Chamberlain, Steven J. Deitz, and Lawrence Snyder. A comparative study of the nas mg benchmark across parallel languages and architectures. In *Supercomputing '00 : Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 46, Washington, DC, USA, 2000. IEEE Computer Society.

[CHC09] Camille Coti, Thomas Herault, and Franck Cappello. MPI applications on grids : a topology-aware approach. In LNCS, editor, *Proceedings of the 15th European Conference on Parallel and Distributed Computing (EuroPar'09)*, Delft, the Netherlands, August 2009.

[CHP+08] Camille Coti, Thomas Herault, Sylvain Peyronnet, Ala Rezmerita, and Franck Cappello. Grid services for MPI. In ACM/IEEE, editor, *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'08)*, pages 417–424, Lyon, France, May 2008.

[FHG04] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. *RFC2582*, 2004.

[FK97] Ian Foster and Carl Kesselman. Globus : A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2) :115–128, Summer 1997.

[FK99] Ian Foster and Carl Kesselman. *The Grid : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.

[Flo03] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649 (Experimental), December 2003.

[Fos02] Ian Foster. What is the Grid ? A Three Point Checklist. *GRIDToday*, July 2002.

[FRS00] Ian Foster, Alain Roy, and Volker Sander. A quality of service architecture that combines resource reservation and application adaptation. In *8th International Workshop on Quality of Service*, pages 181–188, 2000.

[FY02] Ahmad Faraj and Xin Yuan. Communication Characteristics in the NAS Parallel Benchmarks. In *IASTED PDCS*, pages 724–729, 2002.

[GBD+94] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Bob Manchek, and Vaidy Sunderam. *PVM : Parallel Virtual Machine-A User's Guide and Tutorial for Network Parallel Computing*. MIT Press, 1994.

[GFB+04] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI : Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[GHD00] William L. George, John G. Hagedorn, and Judith E. Devaney. IMPI : Making MPI Interoperable. *Journal of Research of the National Institute of Standards and Technology*, 105 :343–428, 2000.

[GHG04] Yunhong Gu, Xinwei Hong, and Robert L. Grossman. Experiences in Design and Implementation of a High Performance Transport Protocol. In *SC '04 : Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 22, Washington, DC, USA, 2004. IEEE Computer Society.

[GJG+05] Fabrizio Gagliardi, Bob Jones, François Grey, Marc-Elian Bégin, and Matti Heikkurinen. Building an infrastructure for scientific Grid computing : status and goals of the EGEE project. *Philosophical Transactions of the Royal Society A : Mathematical, Physical and Engineering Sciences*, pages 1729–1742, 2005.

[GKNpV93] Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Kiem phong Vo. A Technique for Drawing Directed Graphs. *IEEE Transactions on Software Engineering*, 19 :214–230, 1993.

[GLDS96] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. High-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6) :789–828, 1996.

[Gog08] Brice Goglin. Design and Implementation of Open-MX : High-Performance Message Passing over generic Ethernet hardware. In *CAC 2008 : Workshop on Communication Architecture for Clusters, held in conjunction with IPDPS 2008*. IEEE Computer Society Press, April 2008.

[Gro02] William Gropp. MPICH2 : A New Start for MPI Implementations. In *Recent Advances in PVM and MPI : 9th European PVM/MPI Users' Group Meeting*, Linz, Austria, Oct. 2002.

[GRR99] Edgar Gabriel, Michael Resch, and Roland Rühle. Implementing MPI with optimized algorithms for metacomputing. In *Proc. of the Third MPI Developer's and User's Conference (MPIDC'99)*, pages 31–41, 1999.

[GS03] S. Goteti and J. Subhlok. Communication pattern based node selection for shared networks. pages 69–76, June 2003.

[Gui09] Romaric Guillier. *Methodologies and Tools for the Evaluation of Transport Protocols in the Context of Highspeed Networks*. PhD thesis, ENS Lyon, Université de Lyon, October 2009.

[KBV00] Thilo Kielmann, Henri E. Bal, and Kees Verstoep. Fast measurement of logp parameters for message passing platforms. In *IPDPS '00 : Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, pages 1176–1183, London, UK, 2000. Springer-Verlag.

[Kel03] Tom Kelly. Scalable TCP : Improving Performance in Highspeed Wide Area Networks. *SIGCOMM Comput. Commun. Rev.*, 33(2) :83–91, 2003.

[KHB+99] Thilo Kielmann, Rutger F.H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A.F. Bhoedjang. MagPIe : MPI's Collective Communication Operations for Clustered Wide Area Systems. In *Proc. Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'99)*, pages 131–140, Atlanta, GA,, May 1999.

[KHR02] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *SIGCOMM '02 : Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 89–102, New York, NY, USA, 2002. ACM.

[KKFT02] S. Kopparty, S.V. Krishnamurthy, M. Faloutsos, and S.K. Tripathi. Split TCP for mobile ad hoc networks. *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, 1 :138–142, Nov. 2002.

[KSF+02] Nicholas T. Karonis, Bronis de Supinski, Ian T. Foster, William Gropp, and Ewing L. Lusk. A Multilevel Approach to Topology-Aware Collective Operations in Computational Grids. *CoRR*, cs.DC/0206038, 2002.

[LBS06] S. Liu, T. Basar, and R. Srikant. TCP-Illinois : A Loss and Delay-Based Congestion Control Algorithm for High-Speed Networks. In *VALUETOOL*, October 2006.

[LM97] T. V. Lakshman and Upamanyu Madhow. The performance of tcp/ip for networks with high bandwidth-delay products and random loss. *IEEE/ACM Trans. Netw.*, 5(3) :336–350, 1997.

[MHR03] M. Mathis, J Heffner, and R Reddy. Web100 : Extended TCP Instrumentation for Research, Education and Diagnosis. *ACM Computer Communications Review*, 33(3), July 2003.

[Miu06] Kenichi Miura. Overview of Japanese science Grid project NAREGI. *Progress in Informatics*, Volume 3, 2006.

[MKK+05] M Matsuda, T Kudoh, Y Kodama, R Takano, and Y Ishikawa. TCP Adaptation for MPI on Long-and-Fat Networks. In *Proceedings of Cluster05*, Boston, Massachusetts, USA, September 2005.

[MKK+06] M Matsuda, T Kudoh, Y Kodama, R Takano, and Y Ishikawa. Efficient MPI Collective Operations for Clusters in Long-and-Fast Networks. In *Proceedings of Cluster06*, Barcelona, Spain, Sept. 2006.

[MTM+09] Chao Ma, Yong Meng Teo, Verdi March, Naixue Xiong, Ioana Romelia Pop, Yan Xiang He, and Simon See. An approach for matching communication patterns in parallel applications. In *IPDPS '09 : Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–12, Washington, DC, USA, 2009. IEEE Computer Society.

[MX06] Myrinet Express (MX) : A High-Performance, Low-Level, Message-Passing Interface for Myrinet. Technical Report Version 1.2, October 2006.

[NK03] I. Foster N. Karonis, B. Toonen. MPICH-G2 : A Grid-Enabled Implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, pages 551–563, 2003.

[NMSM06] Aroon Nataraj, Allen D. Malony, Sameer Shende, and Alan Morris. Kernel-Level Measurement for Integrated Parallel Performance Views : the KTAU Project. In *Cluster Computing*, pages 1–12, September 2006.

[PB98] Sundeep Prakash and Rajive L. Bagrodia. MPI-Sim : Using Parallel Simulation To Evaluate Mpi Programs. In *Proceedings of the 1998 Winter Simulation Conference - WSC '98*, 1998.

[PJ04] A. Pant and H. Jafri. Communicating efficiently on cluster based grids with mpich-vmi. In *CLUSTER '04 : Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 23–33, Washington, DC, USA, 2004. IEEE Computer Society.

[PP02] Scott Pakin and Avneesh Pant. VMI 2.0 : A Dynamically Reconfigurable Messaging Layer for Availability, Usability, and Management. In *Proceedings of the The 8th International Symposium on High Performance Computer Architecture (HPCA-8), Workshop on Novel Uses of System Area Networks (SAN-1)*, 2002.

[PSB03] Martin Poeppe, Silke Schuch, and Thomas Bemmerl. A message passing interface library for inhomogeneous coupled clusters. In *IPDPS '03 : Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 199.2, Washington, DC, USA, 2003. IEEE Computer Society.

[Rab04] Rolf Rabenseifner. Optimization of collective reduction operations. In *Computational Science - ICCS 2004*, pages 1–9, 2004. http://www.springerlink.com/content/hha38fla0p0nhp1x.

[Rez09] Ala Rezmerita. *Contribution aux intergiciels et protocoles pour les grappes virtuelles*. PhD thesis, Université Paris-Sud XI, Septembre 2009.

[RFG+00] Alain J. Roy, Ian Foster, William Gropp, Brian Toonen, Nicholas Karonis, and Volker Sander. MPICH-GQ : quality-of-service for message passing programs. In *Supercomputing '00 : Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, page 19, Washington, DC, USA, 2000. IEEE Computer Society.

[RS04] Alain Roy and Volker Sander. *GARA : a uniform quality of service architecture*, pages 377–394. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[RX05] Injong Rhee and Lisong Xu. Cubic : a new tcp-friendly high-speed tcp variant. In *Third International Workshop on Protocols for Fast Long-Distance Networks*, February 2005.

[RZR98] Luiz De Rose, Ying Zhang, and Daniel A. Reed. SvPablo : A Multi-Language Performance Analysis System. In *In 10th International Conference on Performance Tools*, pages 352–355, 1998.

[SG09] Sebastien Soudan and Romaric Guillier. TCP Raw, Sept. 2009.

[SHON08] Lucas Mello Schnorr, Guillaume Huard, Philippe Olivier, and Alexandre Navaux. 3D Approach to the Visualization of Parallel Applications and Grid Monitoring Information. In *The 9th IEEE/ACM International Conference on Grid Computing*. IEEE, 2008.

[SJM06] S. Shao, A.K. Jones, and R. Melhem. A compiler-based communication analysis approach for multiprocessor systems. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 10 pp.–, April 2006.

[SL04] R.N. Shorten and Doug Leith. H-TCP : TCP for high-speed and long-distance networks. In *Proceedings of 2nd International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet'04)*, Argonne, Illinois USA, feb. 2004.

[SM06] Sameer S. Shende and Allen D. Malony. The Tau Parallel Performance System. *International Journal of High Performance Computing Application*, 20(2) :287–311, 2006.

[SMM98] Jeffrey Semke, Jamshid Mahdavi, and Matthew Mathis. Automatic TCP buffer tuning. In *Proceedings of the ACM SIGCOMM '98*, pages 315–323, New York, NY, USA, 1998. ACM Press.

[Swa04] Martin Swany. Improving Throughput for Grid Applications with Network Logistics. In *SC '04 : Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 23, Washington, DC, USA, 2004. IEEE Computer Society.

[TKK+05] Ryousei Takano, Tomohiro Kudoh, Yuetsu Kodama, Motohiko Matsuda, H. Tezuka, and Yutaka Ishikawa. Design and Evaluation of Precise Software Pacing Mechanisms for Fast Long-Distance Networks. In *3rd Intl. Workshop on Protocols for Fast Long-Distance Networks (PFLDnet05)*, 2005.

[TMK+07] Ryousei Takano, Motohiko Matsuda, Tomohiro Kudoh, Yuestu Kodama, F. Okazaki, and Yutaka Ishikawa. Effects of Packet Pacing for MPI Programs in a Grid Environment. In *Cluster2007*, 2007.

[TMK+08] Ryousei Takano, Motohiko Matsuda, Tomohiro Kudoh, Yuetsu Kodama, Fumihiro Okazaki, Yutaka Ishikawa, and Yasufumi Yoshizawa. High Performance Relay Mechanism for MPI Communication Libraries Run on Multiple Private IP Address Clusters. In *CCGrid*, pages 401–408, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[TSZS06] K. Tan, Jingmin Song, Qian Zhang, and Murari Sridharan. A Compound TCP Approach for High-speed and Long Distance Networks. In *IEEE INFOCOM*, April 2006.

[VKM02] D. Velenis, D. Kalogeras, and B. Maglaris. SaTPEP : a TCP Performance Enhancing Proxy for Satellite Links. In *2nd International IFIPTC6 Networking Conference*, May 2002.

[XHR04] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary increase congestion control for fast long-distance networks. In *INFOCOM*, 2004.

Sites Web

[DAS] DAS-3 : Distributed ASCI Supercomputer 3. http://www.cs.vu.nl/das3/.

[Gri] GridMPI Project. http ://www.gridmpi.org/gridmpi.jsp.

[IMB] Intel MPI Benchmark (IMB). http://www.intel.com/cd/software/products/asmo-na/eng/cluster/mpi/219847.htm.

[NS2] The Network Simulator - NS-2. http://www.isi.edu/nsnam/ns/index.html.

[Pro] Description du mode de communication de cg. http://proactive.inria.fr/index.php?page=nas_benchmarks .

[Sita] Site RENATER. Renater5. http://www.renater.fr/.

[Sitb] Site Web de Infiniband. http://www.infinibandta.org/.

[Sitc] Site Web de Myricom. http://www.myri.com/.

[Sitd] Site Web de Quadrics. http://www.quadrics.com/.

[TGQ+] A. Tirumala, M. Gates, F. Qin, J. Dugan, and J. Ferguson. Iperf - The TCP/UDP bandwidth measurement tool. http://dast.nlanr.net/Projects/Iperf.

Standards et RFCs

[APS99] M. Allman, V. Paxson, and W. Stevens. RFC 2581 : TCP Congestion Control. *RFC2581*, April 1999. http://www.isi.edu/in-notes/rfc2581.txt.

[BBC+98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. *RFC2475*, December 1998. http ://www.ietf.org/rfc/rfc2475.txt.

[BKG+01] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby. Performance enhancing proxies intended to mitigate link-related degradations. *RFC3135*, June 2001. http ://www.isi.edu/in-notes/rfc3135.txt.

[Ins81] Information Science Institute. Transmission control protocol. *RFC793*, September 1981. http://tools.ietf.org/html/rfc793.

[MPI94] MPI standard 1.0, Juin 1994. http://www.mpi-forum.org/ .

[MPI97] MPI standard 2.0, Juillet 1997. http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html .

[MPI08] MPI standard 1.3, Juillet 2008. http://www.mpi-forum.org/docs/mpi-1.3/mpi-report-1.3-2008-05-30.pdf .

[MPI09] MPI standard 2.2, Septembre 2009. http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf .

[NBBB98] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers. *RFC2474*, December 1998. http ://www.ietf.org/rfc/rfc2474.txt.

[Ope] OpenMP API specification. http://www.openmp.org/mp-documents/spec30.pdf.

[Ste07] Randall R. Stewart. RFC 4960 : Stream Control Transmission Protocol. *RFC4960*, September 2007. http ://tools.ietf.org/html/rfc4960.

Publications

Conférences internationales

[GHK+07] Romaric Guillier, Ludovic Hablot, Yuetsu Kodama, Tomohiro Kudoh, Fumihiro Okazaki, Ryousei Takano, Pascale Vicat-Blanc Primet, and Sebastien Soudan. A study of large flow interactions in high-speed shared networks with Grid5000 and GtrcNET-1. In *PFLDnet 2007*, February 2007.

[HGM+07] Ludovic Hablot, Olivier Glück, Jean-Christophe Mignot, Stéphane Genaud, and Pascale Vicat-Blanc Primet. Comparison and tuning of MPI implementations in a grid context. In *In Proceedings of 2007 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 458–463, September 2007.

[SGH+07] Sebastien Soudan, Romaric Guillier, Ludovic Hablot, Yuetsu Kodama, Tomohiro Kudoh, Fumihiro Okazaki, Ryousei Takano, and Pascale Vicat-Blanc Primet. Investigation of ethernet switches behavior in presence of contending flows at very high-speed. In *PFLDnet 2007*, February 2007.

Conférences nationales

[HGMVBP08] Ludovic Hablot, Olivier Glück, Jean-Christophe Mignot, and Pascale Vicat-Blanc Primet. Etude d'implémentations mpi dans une grille de calcul. In *Actes de Renpar'08*, Février 2008.

Rapports de recherche

[GHK+06] Romaric Guillier, Ludovic Hablot, Yuetsu Kodama, Tomohiro Kudoh, Fumihiro Okazaki, Pascale Vicat-Blanc Primet, Sebastien Soudan, and Ryousei Takano. A study of large flow interactions in high-speed shared networks with Grid5000 and GtrcNET-10 instruments. Research Report 6034, INRIA, 11 2006. Also available as LIP Research Report RR2006-43.

[GHVBP07] Romaric Guillier, Ludovic Hablot, and Pascale Vicat-Blanc Primet. Towards a User-Oriented Benchmark for Transport Protocols Comparison in very High Speed Networks. Research Report 6244, INRIA, 07 2007. Also available as LIP Research Report RR2007-35.

[GHVBPS06] Romaric Guillier, Ludovic Hablot, Pascale Vicat-Blanc Primet, and Sebastien Soudan. Evaluation des liens 10 GbE de Grid'5000. Research Report 6047, INRIA, 12 2006.

[HGM+07] Ludovic Hablot, Olivier Glück, Jean-Christophe Mignot, Stéphane Genaud, and Pascale Vicat-Blanc Primet. Comparison and tuning of MPI implementations in a grid context. Research Report 6200, INRIA, 05 2007.

[HGM+09] Ludovic Hablot, Olivier Glück, Jean-Christophe Mignot, Romaric Guillier, Sébastien Soudan, and Pascale Vicat-Blanc Primet. Interaction between MPI and TCP in grids. Research Report RR-6945, INRIA Rhône-Alpes, LIP, ENS Lyon, 2009.

[SGH+06] Sebastien Soudan, Romaric Guillier, Ludovic Hablot, Yuetsu Kodama, Tomohiro Kudoh, Fumihiro Okazaki, Pascale Vicat-Blanc Primet, and Ryousei Takano. Investigation of ethernet switches behavior in presence of contending flows at very high-speed. Research Report 6031, INRIA, 11 2006. Also available as LIP Research Report RR2006-38.