Interaction between MPI and TCP in grids.

Ludovic Hablot, Olivier Glück, Jean-Christophe Mignot, Romaric Guillier, Sébastien Soudan, Pascale Vicat-Blanc Primet

Laboratoire de l'Informatique du Parallélisme

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

Interaction between MPI and TCP in grids

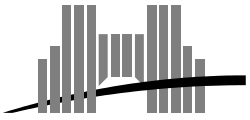
Ludovic Hablot,
Olivier Glück,
Jean-Christophe Mignot,

Romaric Guillier,
Sébastien Soudan,
Pascale Primet

Mai 2008

Research Report N° RRLIP2009-20

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

Interaction between MPI and TCP in grids

Ludovic Hablot, Olivier Glück, Jean-Christophe Mignot,
Romaric Guillier, Sébastien Soudan, Pascale Primet

Mai 2008

Abstract

As MPI applications are more and more resource consuming, they need to be executed on grids. The communications on the WAN interconnecting clusters mostly use TCP which suffers from WAN features: high latency, sharing between users, bandwidth smaller than the aggregate bandwidth of the nodes.
In this report, we first study the interaction between MPI and TCP on grids. We show why the nature of MPI traffic raises problems while using TCP on WAN links. TCP's loss detection and congestion control mechanism can both slow down the application.
Then, we propose MPI5000, a transparent applicative layer between MPI and TCP, using proxies to improve the execution of MPI applications on a grid. Proxies aim at splitting TCP connections in order to detect losses faster and avoid to return in a slowstart phase after an idle time.
Finally, we test our layer on Grid'5000, the French research grid, using MPICH2. The results on the NPB (NAS Parallel Benchmarks) validate our architecture that reduces the number of idle timeout and the number of long-distance retransmissions for certain benchmarks, namely BT, SP and LU benchmarks. Using MPI5000, these applications can decrease their execution time by 35%, 28%, and, 15% respectively.

Keywords: MPI, TCP, Grid'5000, proxies, TCP Split, MPI5000

Résumé

Comme les applications parallèles telles que les applications MPI nécessitent, il devient nécessaire de les exécuter sur des grilles. Cependant, les communications sur le WAN qui interconnecte les clusters de la grille souffrent des caractèristiques intrinsèques du WAN et de l'utilisation de TCP comme protocole de transport : latence élevée, réseau partagé, bande passante inférieure à la bande passante aggregée des noeuds.
Dans ce rapport, nous étudions dans un premier temps, l'interaction entre MPI et TCP sur les grilles de calcul. Nous montrons que la nature même du trafic MPI pose problème à TCP sur les liens WAN. Les mécanismes de détection d'erreur et de contrôle de congestion de TCP sont à même de ralentir l'application.
Ensuite, nous présentons, MPI5000, une architecture à base de proxys placée de manière transparente entre la couche MPI et la couche TCP, qui permet d'améliorer l'execution d'application MPI sur une grille. Les proxys permettent de diviser les connections TCP afin de détecter les pertes plus rapidement et d'éviter de retourner dans le slowstart après une phase d'inactivité.
Enfin, nous présentons les résultats de nos expériences avec MPICH2 sur Grid5000, la grille de recherche française. Les résultats sur les NPB (Nas Parallel Benchmarks) valident notre approche qui permet de réduire le nombre de retour en slowstart et le nombre de retransmissions pour certains benchmarks tels que BT, SP, et LU. En utilisant MPI5000, ces applications diminuent leur temps d'exécution de respectivement 35%, 28% et 15%.

Mots-clés: MPI, TCP, Grid'5000, proxys, TCP Split, MPI5000

1 Introduction

This paper deals with the execution of parallel applications on grid platforms. Many parallel applications are written with the MPI (Message Passing Interface) library. MPI[19] is a standard that defines communication primitives for parallel applications. It includes both point to point (MPI_Send, MPI_Recv ...) and collective communication functions (like MPI_Gather, MPI_Alltoall...). While applications are well executed on clusters, as they are more and more resource-consuming, they need to be efficiently executed on grids. Many implementations are available for the grid like MPICH2 [9], OpenMPI [6] or MPICH-G2 [20] even if they need tuning to be efficiently executed with TCP in a grid [11].

Grids are a pool of computing resources like nodes or data servers connected together. But from an MPI application's point of view, they should be seen as an interconnection of clusters by a wide area network (WAN). As this WAN is shared by all grid users, applications must take care of concurrent traffic and fairly share the network. This is usually achieved by TCP. WAN bandwidth is usually much smaller than required to prevent congestion if all the nodes of one site send data to another site. It is consequently a bottleneck for the application. In the WAN, the latency is high and thus, costly. The impact of latency can be reduced by avoiding long-distance communications either by doing placement of processes[8] or by providing optimized algorithms for collectives operations (in MagPIe [14] or MPICH-G2 [12]). However, if we are using TCP, the time to detect a loss or repair it, depends on RTT (Round Trip Time) and is therefore more costly on a WAN than on a LAN. Splitting TCP connections can solve this problem [15].

To take these problems into account, we put forward MPI5000, a communication layer between the application (MPI for example) and TCP that can be used automatically and transparently. The idea is to introduce proxies at the LAN/WAN interface in order to:

- give the application the knowledge that the grid is an interconnection of clusters.
- implement the TCP split. Each end-to-end TCP connection (LAN-WAN-LAN) is replaced by 3 connections: LAN-LAN, WAN-WAN, LAN-LAN.
- take decisions and implement optimizations on proxies: bandwidth reservation between proxies, communication scheduling, parallel long-distance connections, use of a modified TCP.

This report studies the advantages of TCP splitting for MPI applications executed on a grid and presents our architecture.

The rest of the report is organized as follows. Section 2 explains what problems are raised by using TCP for MPI communications on long-distance links. Then, Section 3 introduces the advantages of our approach and some implementation details. Section 4 presents the evaluation of MPI5000 on the french research grid, Grid'5000[4]. Section 5 discusses about related work. Finally, we conclude and give some future researches we will work on in Section 6.

2 Grids, MPI, and TCP: interaction overview

MPI applications alternate communication and computation phases. When a computation phase is finished, the application waits for new data to compute. MPI applications communicate with small to medium message size (usually less than 1 MB) and generate a bursty traffic

[22]. These bursts are likely to fill the network equipment queues and generate losses and retransmissions. This ultimately increases the application completion time as the execution flow is usually waiting for the next message to continue its execution.

As explained in the introduction, in grids, losses and retransmissions at TCP level are even more costly due to the high latency, the sharing of the WAN, and the bottleneck at the WAN/LAN interface. In this section, we first present TCP features that have an impact on MPI performances. Then, we analyze how it can raised problems for these applications. Finally, we will point out what should be improved for a more efficient execution of MPI applications on grids.

2.1 TCP features

TCP is a reliable protocol that aims at sharing a link fairly between many connections. In grids, using TCP on a the WAN guarantees a statistical fair bandwidth sharing between all applications. We now detail two TCP mechanisms that impact MPI applications: error control and congestion control.

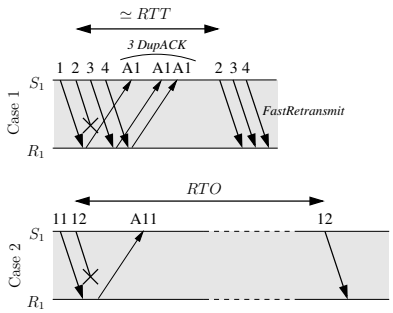


Figure 1: TCP behavior in case of loss

- Error control: This mechanism guarantees loss detection and retransmission. TCP stores the packets in a buffer until they are acknowledged (ACK) by the receiver. If the packet is lost, TCP retransmits it. Figure 1 illustrates the two cases of loss detection:
  - Case 1: TCP waits for 3 duplicate ACKs (DupACKs) of segment 1 to retransmit the lost segment 2 by activating the FastRetransmit algorithm. If there is enough segment to transmit after the loss, the rough estimate for the time before retransmission is one RTT (between 10ms to 100ms).
  - Case 2: if there is nothing to transmit after the loss of message 12, it is necessary to wait for a timeout (RTO) before retransmitting it. This RTO is function of RTT and is much larger than the RTT (usually 200 ms + RTT in Linux)). MPI applications are highly penalized in this case.

- Congestion control: This mechanism aims at sending a controlled amount of data, in order to balance the charge between all connections and avoid congestion. TCP uses a congestion window that determines the amount of data that can be sent at the same time on a link. Figure 2 show the evolution of the congestion window. In order to determine the available bandwidth of a link, TCP first uses a slowstart mechanism. The congestion

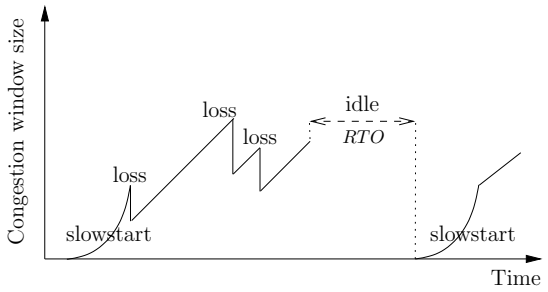


Figure 2: Evolution of TCP congestion window

window is first set at two packets and increases every ACK reception exponentially until a loss occurs. Then, in steady state mode, this window increases linearly. When a loss occurs (detected either by three duplicate ACKs or a Timeout), the congestion window decreases. Finally, after an idle time (nothing is sent on a connection for a long time), TCP enters a slowstart phase again.

2.2 Problems using MPI over TCP on grids

As the WAN link is shared by the application and the LAN/WAN interface is a bottleneck, most losses occur while communicating on the WAN. Since the reception of DupACKs depends on RTT, a loss on the WAN takes a longer time to be detected than on a LAN. In MPI applications, the case 2 of Figure 2 mainly occurs because MPI uses a message paradigm to communicate and consequently sends a small number of successive TCP segments, in contrast with a file transfer for example.

The congestion window limits the amount of data that can be send in the same time. If the congestion window is not large enough, an MPI message can not be sent in one time but have to wait for ACKs to make this window increase and finish its sends. Due to the RTT, this is costly on high latency links. Moreover, as slowstart is activated again after an idle time, if an MPI application computes longer than the idle timeout, it will suffer from the reactivation of the slowstart mechanism.

2.3 Focus on designing long-distance MPI architecture

As shown previously, we can sort the problems in two groups:

- Due to latency, on the grid, the application waits longer for DupACKs and for the increase of the TCP congestion window.
- Due to MPI application communication profile, there are many RTO and many idle time.

Table 1 show some clue on these problems. It present the sum of DupAck, RTO and Idle timeout that occur on long-distance connections while executing BT, FT and LU from the the NPB[3] (NAS Parallel Benchmark) suite. The experiments are performed on two clusters of 8 nodes, interconnected by a 1 Gbps link. The figures were obtain with Web100 as described in Section 4.1. DupAck and RTO occur only in FT while the three applications are impacted by idle timeout.

Application	DupACK	RTO	Idle timeout
BT	1	0	5668
FT	275	20	905
LU	1	0	760

Table 1: Sum of DupACK, RTO, and Idle timeout while executing the NPB in a grid.

In order to solve some of these problems, we put forward in the next section an architecture based on LAN/WAN proxies that enables to split TCP connections, and therefore, on the WAN: to reduce the number of idle timeout, and to faster detect and reduce losses.

3 MPI5000 layer

In order to control and improve MPI communications on grids, we propose MPI5000, a transparent layer to execute MPI applications on grids using proxies. It can be used with any MPI implementation by loading the MPI5000 library (i.e. set the environment LD_PRELOAD variable) and launching the MPI5000 daemons (i.e. proxies).

3.1 Overview of MPI5000

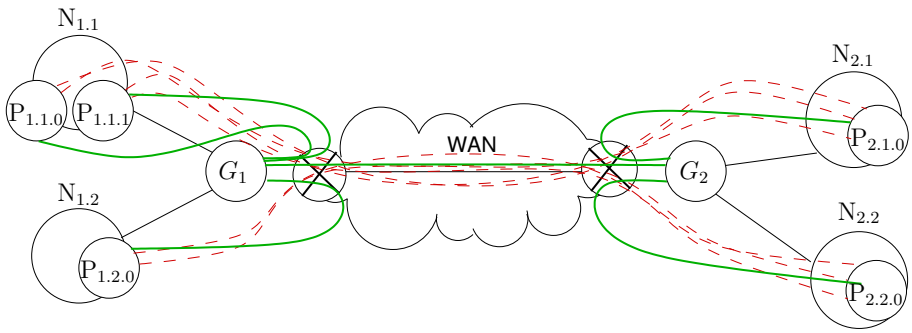


Figure 3: Long distance communications with proxies

Figure 3 illustrates how MPI5000 splits the TCP connections. The dashed red lines on the figure represent the connections without MPI5000 while the plain green lines represent the MPI5000 connections. Each LAN-WAN-LAN connection is replaced by three connections LAN-LAN, WAN-WAN, and LAN-LAN. $N_{s,n}$ is the node n of site s. $P_{s,n,p}$ is the process p executed on node n of site s. G_s is the proxy of site s. Each process is connected to the proxy of its site and proxies are connected together.

Proxies allow both to react faster in case of congestion and to change MPI's bursty traffic into longer flows on long-distance links.

The advantages of our solution are the following and shown on Figure 4 in which we use the notation previously mentionned. $P_{1.1.0}$ and $P_{1.2.0}$ are sender processes from the first site, G_1 and G_2 are proxies from site 1 and site 2 respectively. $P_{2.1.0}$ and $P_{2.2.0}$ are receiver processes, situated on site 2. Gray color represents the WAN. Black or blue arrows represent different connections. Green arrows represent WAN-WAN connections using MPI5000.

- Loss occurs on the LAN (due to a alltoall for example): In case 1 of Figure 4, congestion occurs on a LAN instead of a WAN. Without proxies, if packet 2 is lost, sender $P_{1.1.0}$ waits for three duplicate ACKs generated by packets 3, 4, and 5 and sent by $P_{2.1.0}$, before it knows that the packet is lost and retransmits it. The DupACKs arrive after a long-distance RTT (RTT_w). With proxies, the sender $P_{1.1.0}$, also waits for three DupACKs but it is G_1 that retransmits it. It takes only a local RTT (RTT_l).

  In case 2a, proxies transform a RTO in DupACKs. $P_{1.1.0}$ sends two messages: the first message is composed of packets 2 and 3 and is sent to $P_{2.1.0}$, packets 2' and 3' contain the second message sent to $P_{2.2.0}$. Without proxies if packet 2 is lost and packet 3 is the last packet of a message, $P_{1.1.0}$ will receive only one DupAck and wait for a timeout before retransmitting it. On the contrary, if we use proxies, packets 4 and 5 use the same connection as packets 2 and 3. They contribute to send the DupACKs necessary to do a retransmission that is done by $P_{2.1.0}$ after a local RTT (RTT_l).

- Loss occurs on the WAN (due to cross traffic): in case 2b, $P_{1.1.0}$ sends packets 2 and 3 to $P_{2.1.0}$ while $P_{1.2.0}$ sends packets 2' and 3' to $P_{2.2.0}$. Packet 2 is lost. In the normal case, $P_{1.1.0}$ waits for a timeout before retransmitting packet 2. With proxies, however, at the reception of packets 2' and 3', G_2 sends a DupACK. $P_{1.1.0}$ retransmits packet 2 only after a long-distance RTT (RTT_w).

MPI5000 aims at avoiding timeout on long-distance and local links. It also retransmits DupACKs faster. Therefore, it reduces the waiting time of MPI applications and improve global performances.

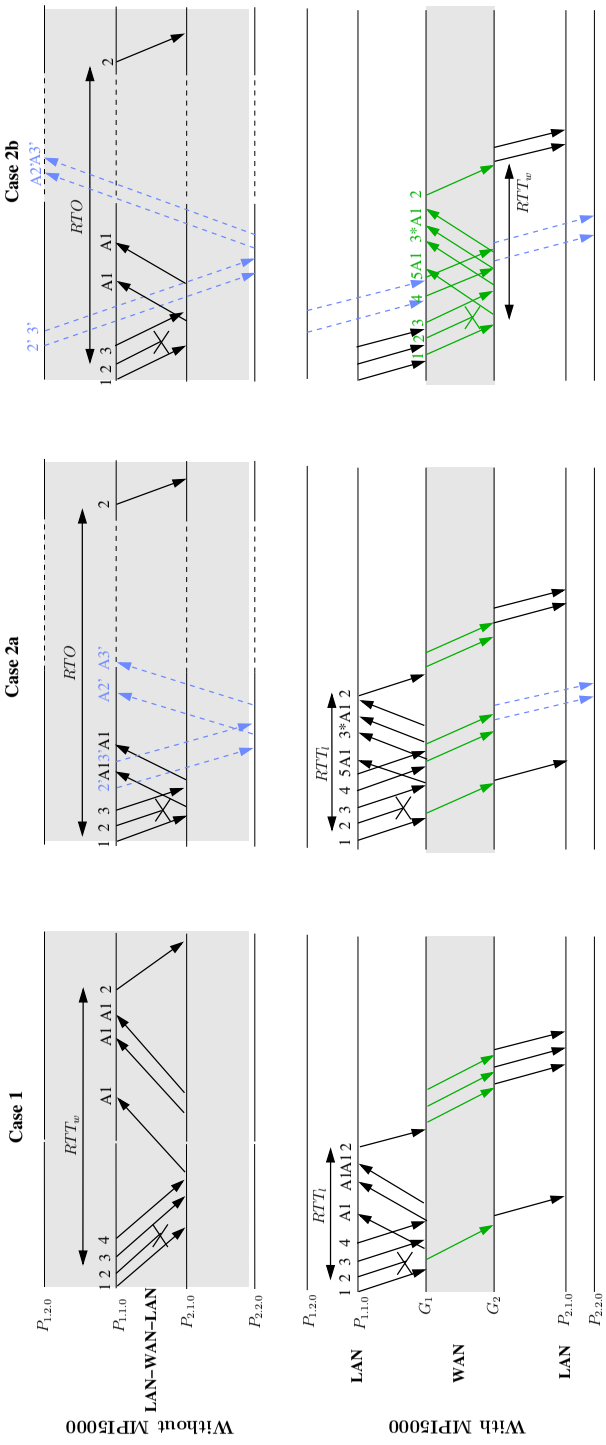


Figure 4: Comparison of TCP without and with proxies.

Our architecture also has an impact on the congestion window. Indeed, proxies help to keep the congestion window closer to the real available bandwidth because they transmit more data than a single process and thus probe the network more regularly. If an application has communication phases longer than the idle timeout but all processes do not communicate synchronously, the proxies help to not go back in slowstart phase because other processes keep the congestion window open.

3.2 MPI5000 implementation

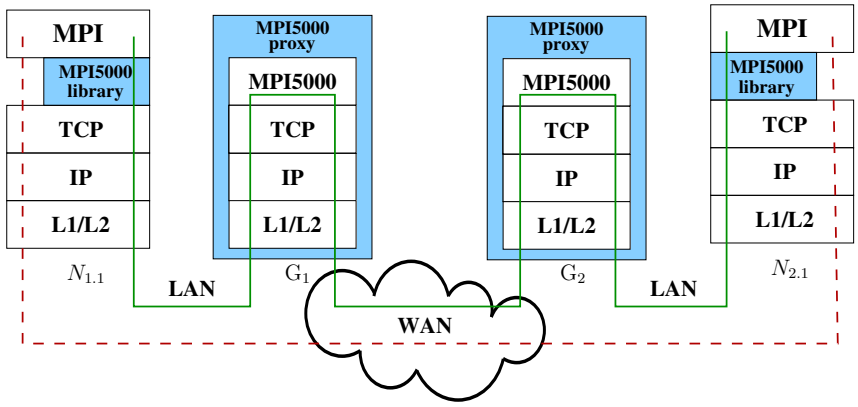


Figure 5: MPI5000 architecture

As shown on Figure 5, MPI5000 is based on two components: a library on node and a daemon program on proxies. The dashed red lines represent the data path from one node to another without MPI5000 while the plain green lines represent it with MPI5000.

In order to route messages between nodes and proxies, we add a header to the MPI message (shown on Figure 6). The header contains a flag that identifies the message type (data or connection), the destination's id, the size of the MPI message and the source's id. An id is described using three numbers: s, the site's number, n, the node's number and p, the process's number on the node.

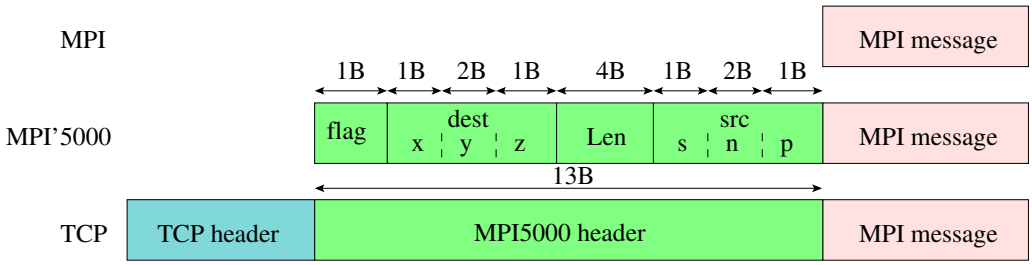


Figure 6: MPI5000 header

3.2.1 MPI5000 library

The library is placed below MPI and can be used with few modifications to the usual MPI execution procedure The mpirun command line is modified in order to call the MPI5000

library (for example env LD_PRELOAD in mpich). The MPI5000 library intercepts functions of socket API (`bind, accept, connect, write/v, read/v, close`) – in other words, we force the MPI program to call our library's functions instead of libc's – and adapt them to MPI5000's architecture. With this mechanism, we are able to use our layer with any MPI implementation. On a `bind()` the library connects the node to the proxy. When a `connect()` is intercepted, the library creates a message with the connection flag set and sends it to the proxy. On a `accept()`, it just waits to the adequate `connect()`.

3.2.2 MPI5000 proxies

The second component of MPI5000 is the proxies. For the moment, they are launched manually but it could be done automatically. The proxies just wait for data and forward it to either another proxy or to local nodes, using the information included in the header. If the connection flag is set, the proxy establishes the connection to the matching node in order to free the pending `accept()`. Once this is done, this connection is closed and only the connection to the proxy remains.

4 Experimental evaluation

This section evaluates the architecture described in previous section and based on the advantages discussed in Section 2.

4.1 Experimental testbed

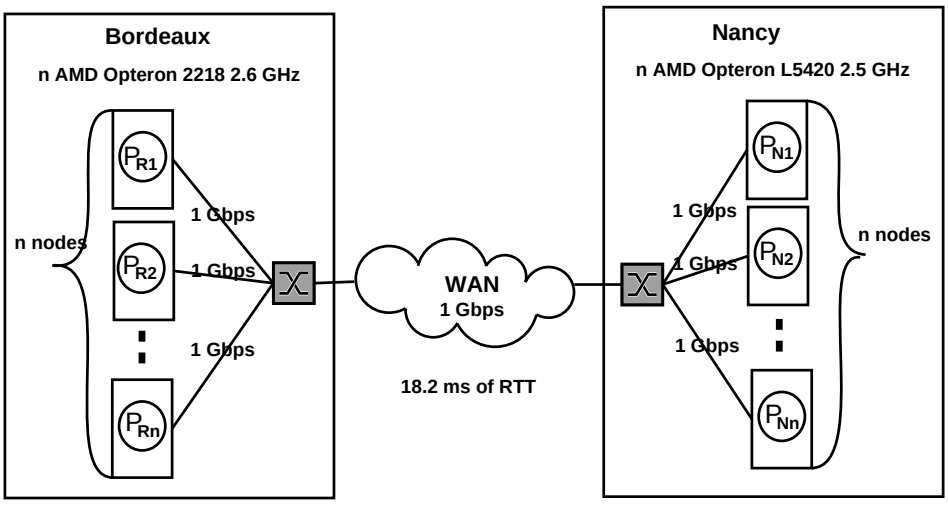


Figure 7: Experimental testbed

Our experiments are conducted on Grid'5000[4], which links nine sites in France, having from 5 ms to 21 ms of RTT. Sites are connected by a dedicated WAN operated by RENATER at 1 or 10 Gbps. This architecture provides researchers a full reconfigurability feature to dynamically deploy and configure any OS on any host. This feature allows them to have administrator rights, to change TCP parameters for instance. Figure 7 shows the experimental testbed used. Bordeaux's nodes are AMD Opteron 2218 2.6 GHz. Nancy's nodes are AMD

Opteron 5420 2.5 GHz connected to the switch by a 1 Gbps Ethernet card. Nancy's and Bordeaux's switches are HP ProCurve 3400cl and HP ProCurve 5406zl respectively. The kernel scheduler frequency is set to 1000Hz. Bordeaux and Nancy are connected by a 1 Gbps link, with 18.2ms of RTT. n, the number of nodes used within a cluster depends of the experiment (1 in case of pingpong, 8 in the NPB case).

TCP uses BIC congestion control, with SACK. In order to reset TCP values (buffer sizes, congestion window...) between two experiments, we disable the save of metrics feature. TCP default receive's and send's buffer sizes are set to 4 MB which correspond to bandwith delay product (BDP, as advised in [24]).

We use a kernel 2.6.28.7 with the Web100[17] 2.5.23 patch applied. Web100 instruments the code of the TCP kernel stack in order to give a view of the TCP behavior. Almost all useful values of a TCP connection like the congestion window size, the number of DupACK, the number of RTO, the RTT, etc... are logged. Each variable is updated continously but we pick up values every only 10 ms.

Cross traffic generation: During some experiments (specified later), we add cross traffic on the long distance link. This cross traffic is carried out with two iperf TCP flows at 1 Gbps on four extra nodes. One flow is sending from one Nancy's node to a Bordeaux's node, another is going the opposite way on different nodes. This traffic congest the long-distance link between Bordeaux and Nancy.

4.2 Proxies impact on a simple pingpong

In this section, we measure the overhead due to the proxies on the execution of simple MPI pingpong.

	MPICH2	MPI5000	Overhead
Latency (μs)	9114	9255	141 μs (1.5%)
Bandwidth ($Mbps$)	840	785	15%

Table 2: MPI latency and bandwidth between the two clusters.

As shown in Table 2, latency is increased by 141 μs for 1 B messages. The overhead of one proxy is the sum of the time from the switch to the proxy and the time to the crossing of TCP stacks. For each proxy, we add four extra copies, from the card to the TCP buffer, from the TCP buffer to the proxy application and the same backwards. While executing a simple pingpong between two nodes of the same cluster, the number of copies and the RTT are equal to the overhead introduced by proxies. Doing this experiment, the round-trip latency is 142μs on the LAN, which is similar to the overhead previously mentionned. Bandwidth decreased from 840 Mbps to 785 Mbps for 33 MB messages, about 15%. Indeed, a higher message size, increases the overhead to do extra copies.

4.3 Performances on NPB

The next experiments use the Nas Parallel Benchmark (NPB[3]) to show the performance of MPI5000. The NPB are a set of eight programs (BT, CG, EP, FT, IS, LU, MG and SP) that have been designed to compare performances of supercomputers but are now also used to compare MPI implementations. The NPB give a good panel of the different parallel

	Comm. type	Quantity of data	Long-distance writes: Size and quantity	Number of long-d. conn.	Execution time on a cluster
BT	P. to Point	2.8 GB	9648 writes of 26 kB + 16112 w. of 160 kB	32	151 s
CG	P. to Point	2.37 GB	15800 writes of 150 kB	8	52 s
FT	Collective	5.9 GB	600 writes < 200 B + 2816 writes > 2 MB	240	s
IS	Collective	0.74 GB	1151 writes < 400 B + 0.5 MB < 1400 w. < 0.6 MB	240	10 s
LU	P. to Point	0.62 GB	200000 writes of 1 kB + 2000 w. of 200kB	8	72 s
MG	P. to Point	0.19 GB	8842 * diff. sizes B from 40 B to 135 k	24	6 s
SP	P. to Point	5.1 GB	45 kB<19248 writes<54 kB + 100 kB<32112 w.<160 kB	32	183 s

Table 3: NPB communication features on long-distance with MPICH.

applications that could be executed on a cluster or a grid. Table 3 summarizes the long-distance communication features of NPB 2.4 for B class problem on 16 nodes. We obtain these figures by logging each TCP write size during one NPB execution. We do not care about EP because it mostly computes and does few communications. FT uses the primitive MPI_Alltoall and IS uses MPI_Allreduce and MPI_Alltoallv.

4.3.1 Overhead of MPI5000 on big messages

We run each NPB three times and take the mean execution time. The Figure 8 shows the relative execution time between MPICH2 and MPICH2 with MPI5000 of each NPB. FT and IS show very bad performances with our layer. As shown in Table 3, FT and IS use collective operations of big size. The overhead of the copies in MPI5000 proxies is important on big messages, especially when it is collective communications because all messages are sent synchronously. For example, we run a alltoall of 2 MB on 16 nodes with and without MPI5000. The resulting relative completion time is 2.74 which is similar to the time observed for FT on Figure8 (2.86). The same observation can be done with IS, but message sizes are smaller, and so is the overhead. Thus, if the proxies can not forward data fast enough, their overhead is high.

4.3.2 Impact on idle timeout

NAS	MPICH2 without MPI5000	MPICH2 with MPI5000
BT	331	323
CG	725	427
LU	185	179
MG	73	70
SP	487	426

Table 4: Number of idle timeouts in the NPB with and without MPI5000

The number of congestion signals (DupACK and RTO) are obtained thanks to the Web100 patch. Table 4 shows the number of time the congestion window size is decreased without loss

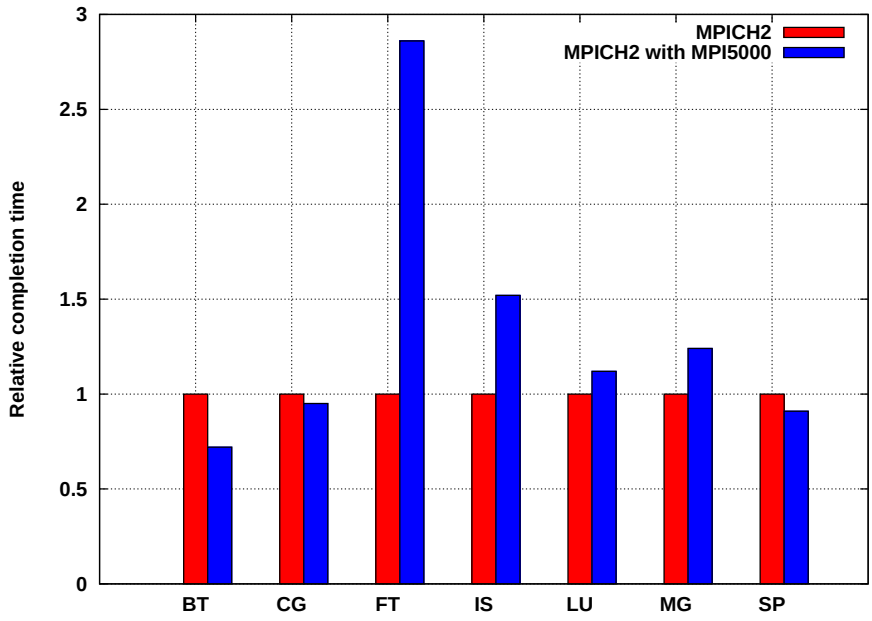


Figure 8: Relative performance of NPB normalized to MPICH2

signal i.e. the number of idle timeout we have measured with and without MPI5000 in the NPB. The figures for MPICH2 without MPI5000 are a mean on all long-distance connections while for MPICH2 with MPI5000 they are taken on the proxy long-distance connection. All NAS show a smaller number of idle timeout with MPI5000.

	MPICH2 without MPI5000		MPICH2 with MPI5000
NAS	Execution time (s)	Execution time without slowstart after idle (s)	Execution time (s)
BT	204	171	147
CG	122	122	116
LU	66	71	74
MG	11	11	14
SP	242	203	221

Table 5: Comparison of NPB execution time with and without slowstart after idle

In order to confirm these results, we disable the slowstart after idle TCP feature (an option in linux). The results are shown in Table 5. CG, LU and MG with the slowstart disabled show a similar completion time with or without the slowstart after idle. Thus, reducing the number of idle timeout can not improve performance. However, in BT and SP, disabling slowstart after idle improve completion time. These results confirm that MPI5000 reduces the number of idle timeouts.

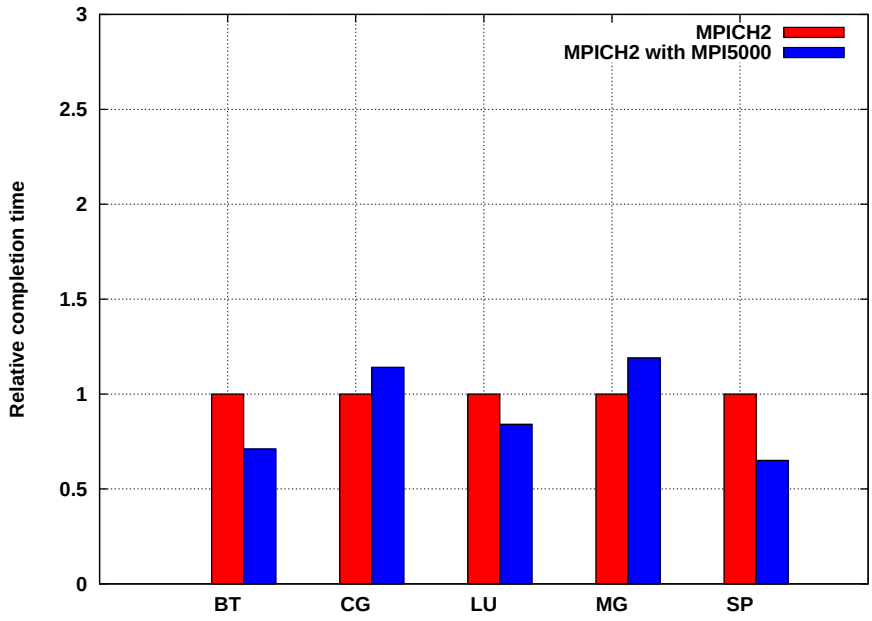


Figure 9: Relative completion time of NPB normalized to MPICH2 with cross-traffic

4.3.3 Impact on RTO and DupACK

The previous experiments do not explain the impact of MPI5000 on CG, LU and MG. Moreover, the Web100 results on these NPB show that there is no loss signals (neither duplicate ACK or RTO). In order to see what can MPI5000 improve under congestion conditions, we add cross-traffic, generated as previously described in Section 4.1.

The Figure 9 show that BT, LU, and, SP benefit from MPI5000 in case of cross-traffic. However, CG decrease its performance compare to the case without cross traffic. This is probably due to a congestion window that is not increased fast enough but further investigation is needed on this point.

The Table 6 shows the number of congestion signals with and without MPI5000 in case of cross-traffic. In the MPI5000's case, local refers to connections from nodes to proxies and distant to the connection between the proxies.

BT, LU, MG and SP results show that MPI5000 delete a lot oflong distance RTO and DupACKs. However, congestion on MG is not high enough to overlap the overhead introduced by proxies. Its relative completion time is reduced compare to the case without cross-traffic. SP is even better in this case than without cross-traffic and shows an improving completion time by 35%.

5 Related work

Many MPI implementations are available for grids. While some of them manage heterogeneity of interconnects, others like MPICH2[9] or GridMPI[18] also propose optimized collec-

	MPICH2		MPI5000			
	Distant		Local		Distant	
NAS	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

Table 6: DupACK and RTO with and without MPI5000 in case of cross-traffic

tive operations to communicate efficiently between clusters. For example in MPICH2, the MPI_Gather implementation packs all messages from a cluster in one (depending of message size) to send it over a long-distance link. Other work try to optimize task placement to avoid long distance communications [8]. However, this approach needs to know the communication scheme of the application.

As previously explained, MPI traffic is mainly composed of bursts. Some mechanisms like pacing ([2] and [23]) have been proposed to avoid these bursts by spacing packets, thus reducing the number of retransmissions. The pacing mechanism consists in introducing a delay between packets in order to avoid them to be dropped by switches. MPI5000 behaves like pacing in smoothing TCP long-distance traffic.

Other authors propose to use proxies to improve performance like in [5]. The same mechanism, called TCP splitting [15] is used in wireless networks. It aims at putting an interface between a lossy link like satellite or wireless connection and improve TCP performance. We show in this report that it is also useful for wired connections. In [25], the authors propose a layer named *Logistical Session Layer or LSL* between the application and TCP that use some depots on a network path to buffer packet and improve performance. Our work follows a similar approach – their depots are substituted by gateways –, but we specifically focus on MPI applications and their interaction with TCP.

GridMPI's developpers proposed a similar mechanism of proxies using the IMPI protocol to communicate between nodes and proxies [26]. The main purpose of their implementation is to communicate between private clusters. MetaMPICH[21] aims at managing heterogeneity of node communication interfaces and put proxies at the WAN/LAN interface. However, both GridMPI and MetaMPICH proxies are part of the implementation and can not be used with another one. PACX-MPI[7] also use proxies to managed heterogeneity. Their implementation is placed above both MPI and TCP. Therefore, it can not split TCP connections as presented in this article.

SOCKS[16] is a protocol to communicate between a client and a server via a proxy. Tsocks[1], a transparent SOCKS proxying library, uses the same system of hijacking `libc` calls than MPI5000. We could have modified it to adapt it to our architecture. However, we would have to implements a SOCKS proxy calling another SOCKS proxy. Moreover, as SOCKS use IP to forward data, we would have lost the site, node and process information necessary to reduce the number of connections.

Some other protocols have been proposed to avoid drawbacks of TCP on grids. UDT [10] is a protocol based on UDP, to share more efficiently than TCP the bandwidth between a small number of sources doing bulk transfers. The flow control is done using a rate control, a window control, and a bandwidth estimation technique. Our context is clearly different

because messages are small and sent over a lot of connections. However, their solution may be useful for the long-distance connections of MPI5000. XCP [13] proposes to improve the detection of losses by providing active switches. These switches warn the sender when they are dropping a packet due to a congestion. Such an architecture could help to faster detect losses on long-distance links but this architecture is not deployed yet.

6 Conclusion

The execution of MPI applications on grids faces new problems. Grids are an interconnection of clusters linked with a WAN. This network has a high latency where the time to detect and repair losses is costly. The WAN is shared between all grid users, and thus, the use of TCP is necessary to fairly share the bandwidth. Finally, its bandwidth can be a bottleneck if a lot of nodes want to communicate on the WAN at the time. To take these problems into account, we propose to put a proxy at the LAN/WAN interface to: give the application a vision of the grid, split TCP connections, take decisions on this proxy in order to better manage the long-distance connections.

We first show in this report how MPI deals with TCP on grids. MPI waits for messages to continue its execution. Thus, any delay in the reception can potentially slowdown the execution. The loss detection of TCP error control, is done by the reception of duplicate ACKs (which depends of RTT) or an RTO. TCP congestion control is done by a congestion window that limits the message size sent in one time.

Then, we propose MPI5000, a transparent layer that alleviates TCP's drawbacks by adding proxies at the LAN/WAN interfaces. MPI5000 splits each TCP LAN-WAN-LAN connections in three connections LAN-LAN, WAN-WAN and LAN-LAN. This allows to detect losses faster: because the loss occurs on the WAN instead of the LAN or because all the nodes use the same connection and contribute to avoid RTOs. MPI5000 also helps to avoid the slowstart phase after an idle time (time without communications on a link).

Finally, we test MPI5000 on the french research grid, Grid'5000. We show on the execution on NPB that MPI5000 can increase performance. First, we have seen that FT and IS suffer on MPI5000 because the cost of copies on gateways is too important. Second, BT and SP benefit from the reduction of idle timeout. Third, due to the few number of losses during the first experiments, we add cross-traffic on the link to emulate a production grid with a lot of applications sharing the WAN. In this case, BT, LU, and SP show improvements with MPI5000: the number of long-distance DupACK and RTO is reduced and execution time decreases. In conclusion, TCP split is a valid approach to execute MPI applications on grids and MPI5000 can improve performance if the messages are not too big which is not the case in most of MPI applications.

In a future work, we will further investigate TCP's behavior with MPI5000 in order to better understand NPB performances, especially for CG. We will particularly pay attention to the evolution of the congestion window in these cases. In order to validate the transparent approach, we will execute MPI5000 with other MPI implementations. In order to reduce the overhead, we could implement a kernel version of the proxies to avoid two copies in user space. Finally, we will implement and evaluate the optimizations previously described: bandwidth reservation between proxies, communication scheduling, parallel long-distance connections, use of a modified TCP.

References

[1] Tsocks, a transparent SOCKS proxying library. `http://tsocks.sourceforge.net/index.php`.

[2] A. Aggarwal, S. Savage, and T. Anderson. Understanding the Performance of TCP Pacing. *IEEE INFOCOM*, 3:1157–1165, 2000.

[3] 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.

[4] 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/`.

[5] 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`.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] Nicholas T. Karonis, Bronis R. 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.

[13] 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.

[14] 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.

[15] 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.

[16] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. SOCKS Protocol Version 5. RFC 1928 (), 1996.

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

[18] 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.

[19] MPI standard. http://www.mpi-forum.org/.

[20] 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.

[21] 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*, Washington, DC, USA, 2003. IEEE Computer Society.

[22] 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.

[23] Y.Kodama M.Matsuda H.Tezuka R.Takano, T.Kudoh and Y.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*, 2005.

[24] 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.

[25] 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.

[26] 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. pages 401–408, Los Alamitos, CA, USA, 2008. IEEE Computer Society.