HAL
open science

Splitting TCP for MPI Applications Executed on Grids

Olivier Glück, Jean-Christophe Mignot

To cite this version:

Olivier Glück, Jean-Christophe Mignot. Splitting TCP for MPI Applications Executed on Grids. 2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications, May 2011, Busan, South Korea. pp.207-212, 10.1109/ISPA.2011.11 . hal-00767643

HAL Id: hal-00767643

https://inria.hal.science/hal-00767643v1

Submitted on 20 Dec 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Splitting TCP for MPI applications executed on grids

Olivier Glück and Jean-Christophe Mignot
LIP laboratory (Laboratoire de l'Informatique du Parallélisme)
ENS Lyon, INRIA, CNRS, Université de Lyon
69364 Lyon Cedex 07, FRANCE
olivier.gluck@ens-lyon.fr, jean-christophe.mignot@ens-lyon.fr

Abstract

In this paper, we first study the interaction between MPI applications and TCP on grids. Then, we propose MPI5000, a transparent applicative layer between MPI and TCP, using proxies to improve the execution of MPI applications on grids. Proxies aim at splitting TCP connections in order to detect losses faster and avoid to return in a slow-start phase after an idle time. Finally, we evaluate our layer executing the NAS Parallel Benchmarks on Grid'5000, the French research grid. The results show that our architecture reduces the number of idle timeout and of long-distance retransmissions for BT, SP and LU benchmarks. Using MPI5000, these applications can decrease their execution time by 35%, 28%, and, 15% respectively. A comparison with MPICH-G2 performances shows that our layer can even outperform a grid enabled MPI implementation.

1 Introduction

This paper deals with the execution of parallel applications on grid platforms. Many parallel applications are written with the Message Passing Interface (MPI) library. MPI [1] 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.

Grids are a pool of computing resources like nodes or data servers connected together. But from the MPI 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 thanks to TCP. The 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. Moreover, the WAN latency is high and thus, costly. When using TCP, the time to detect a loss or repair it depends on the Round Trip Time (RTT) and is much more costly on a WAN than on a LAN.

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: (1) Give the application the knowledge that the grid is an interconnection of clusters. (2) Implement the TCP split: each end-to-end TCP connection (LAN-WAN-LAN) is replaced by 3 connections: LAN-LAN, WAN-WAN, LAN-LAN. (3) Take decisions and implement optimizations on proxies: bandwidth reservation between proxies, communication scheduling, parallel long-distance connections, use of a modified TCP.

This paper studies the advantages of TCP splitting for MPI applications executed on a grid and presents our architecture. It is organized as follows. Section 2 explains which 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 [2]. Section 5 discusses related works. Finally, we conclude and give some future researches we will work on in Section 6.

2 Interactions between Grids, MPI, and TCP

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

As explained in 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. Most losses occur while communicating on the WAN. Since the RTO timeout and the reception of DupACKs depends on the RTT, a loss on the WAN takes a longer time to be detected than on a LAN.

Moreover, the TCP congestion window limits the amount of data that can be sent at 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, it 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. Thus, TCP congestion control is very costly for MPI applications executed on grids because an idle time may often occur and then highly increases the transfer time of MPI messages.

Thus, there are two kinds of problems: (1) due to a high latency on the grid, the application waits a longer time for DupACKs and for the TCP congestion window increase; (2) due to MPI application communication profile, there might be many RTO timeouts and many idle times.

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

Table 1. Number of DupACKs, RTOs, and Idle timeouts while executing some NPB on grids.

Table 1 shows some clue on these problems. It presents the number of DupAcks, RTOs and Idle timeouts that occur on long-distance connections while executing BT, FT and LU from the NAS Parallel Benchmarks [4] (NAS or NPB) suite. The experiments are performed on Grid'5000. The figures were obtained with Web100, as described in section 4.1. DupAcks and RTOs occur only with FT while the three applications are impacted by idle timeouts.

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 thus to reduce the number of idle timeouts, and to faster detect losses and reduce their number.

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 MPI5000 overview

Figure 1 illustrates how MPI5000 splits the TCP connections. The dashed red lines on the figure represent the

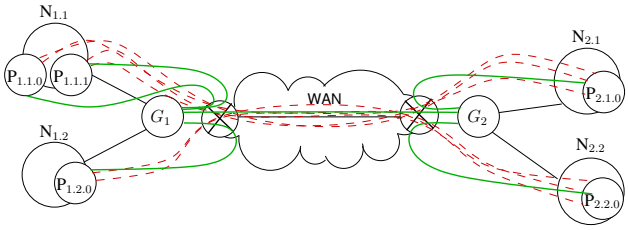

Figure 1. Splitting TCP with MPI5000.

connections without MPI5000 while the plain green lines represent the MPI5000 connections. Thus, with MPI5000, 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 enable both a faster reaction in case of congestion and a transformation of MPI's bursty traffic into longer flows on long-distance links. Indeed, the WAN-WAN MPI5000 connection aggregates the traffic coming from all the processes of the same site.

Thus, MPI5000 eliminates some RTO timeouts on long-distance and local links. It also retransmits DupACKs faster. Therefore, it reduces the TCP waiting time of MPI applications and improve their global performances. Our architecture also has an impact on the congestion window. Indeed, proxies help to keep the congestion window on the WAN connection 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 computation phases on one process longer than the idle timeout but all processes do not communicate synchronously, the proxies avoid to go back in slowstart phase because other processes keep the congestion window widely opened.

3.2 MPI5000 implementation

MPI5000 is based on two components (see Figure 2): a library linked to the application on nodes and a daemon program executed on proxies. The dashed red lines represent the data path from one node to another one without MPI5000 while the plain green lines represent it with MPI5000.

In order to route messages between nodes and proxies, MPI5000 adds a header to the MPI message as shown on Figure 3. 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 (see $P_{s,n,p}$ examples on Figure 1): the site's number (s or x), the node's number (n or y) and the process's number on the node (p or z).

The two components of MPI5000 are now briefly described. The first one, the MPI5000 library, is linked to the

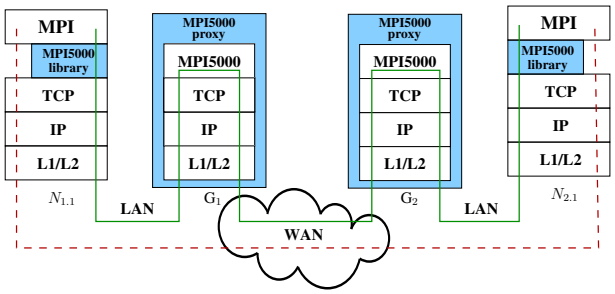

Figure 2. MPI5000 architecture.

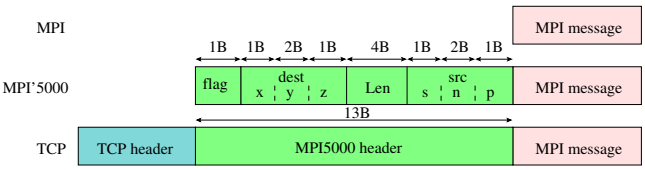

Figure 3. MPI5000 header.

application through the `mpirun` command line (for example `-env LD_PRELOAD` in mpich). Our 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, it is possible to use our layer with any MPI implementation. When a `bind()` is called, the library connects the node to the proxy. When a `connect()` is intercepted, the library creates a MPI5000 message with the connection flag set and sends it to the proxy. When an `accept()` is called, it just waits for the matching `connect()`. The second component of MPI5000 is composed of the proxies. The proxies just wait for data and forward it to either another proxy or to local destination node, 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 evaluations

This section evaluates the architecture described in the previous section and is based on the elements discussed in Section 2. First, we present our experimental testbed. Then, a simple pingpong application is executed with MPI5000 in order to evaluate the raw proxy overhead. Finally, we evaluate MPI5000 performances when executing the NAS Parallel Benchmarks.

4.1 Experimental testbed

Our experiments are conducted on Grid'5000 [2], which connects nine sites in France, with a RTT between 5ms to 21ms. Sites are connected by a dedicated WAN operated by

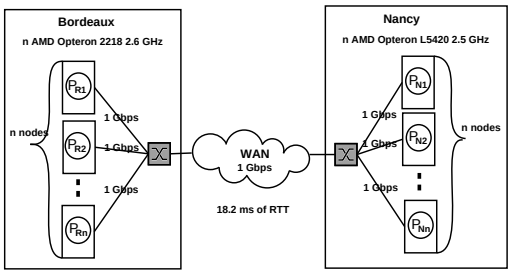

Figure 4. Experimental testbed.

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 4 shows our experimental testbed.

4.2 Proxies impact on a simple pingpong

In this section, we measure the overhead of MPI5000 executing a simple MPI pingpong. With MPI5000, the latency is increased by 141 μs (+1.5%) for 1 Byte messages. This time mainly includes the crossing of two MPI5000 proxies, one on each site. One proxy overhead (see Figure 1 and 2) is the sum of one extra LAN RTT (between the local switch and the proxy node) and the crossing time of proxy communication layers including four extra copies on the proxy (from the network interface card to the TCP buffer, from the TCP buffer to the MPI5000 daemon buffer and the same backwards). Bandwidth is decreased about 7% from 840 Mbps to 785 Mbps for 33 MB messages. Indeed, a higher message size increases the time to do extra copies.

4.3 Nas Parallel Benchmark performances

In this section, we execute the Nas Parallel Benchmarks (NPB [4]) to evaluate the MPI5000 performance using a panel of real applications. First, we present the NAS benchmarks and their communication features. Then, we give the completion time of each benchmark without and with MPI5000 and comment our results. We present further experiments showing that MPI5000 can improve application performances by decreasing the number of idle timeouts and losses on the long-distance link. Last, we compare our results with those of MPICH-G2, a well-known Grid-enabled MPI implementation.

4.3.1 NPB communication features

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 applications that could be executed on a cluster or a grid. We run each NPB three times and take the mean execution time.

	Communication type	Amount of data	Number/size of long-distance writes
BT	Point to Point	2.8 GB	9648 writes of 26 kB + 16112 writes of 160 kB
CG	Point to Point	2.37 GB	15800 writes of 150 KB
FT	Collective	5.9 GB	600 writes<200 B + 2816 writes>2 MB
IS	Collective	0.74 GB	1151 writes<400 B + 0.5 MB<1400 writes<0.6 MB
LU	Point to Point	0.62 GB	200000 writes of 1 KB + 2000 writes of 200 KB
MG	Point to Point	0.19 GB	8842 * diff. sizes from 40 B to 135 KB
SP	Point to Point	5.1 GB	45 KB<19248 writes<54 KB + 100 KB<32112 writes<160 KB

Table 2. NPB communication features on the long-distance link.

		Classification level		
		Low	**Medium**	**High**
Metric	**Comm. frequency**	**>1s** FT, IS	**between 0.1s and 1s** BT, SP	**<0.1s** CG, MG, LU
	Comm. size	**<1KB** LU	**between 1KB and 200KB** BT, SP, CG, MG	**>200KB** FT, IS
	Burstiness	BT, SP, LU	CG, MG	FT, IS

Table 3. Classification of NPB with regards to three communication features.

Table 2 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 on WAN connections during one NPB execution. We do not care about EP because it mostly computes and does very few communications. FT and IS mainly use collective communication primitives: `MPI_Alltoall` for FT, `MPI_Allreduce` and `MPI_Alltoallv` for IS.

In order to understand the following experimental results, Table 3 presents a classification of NPB with regards to three communication metrics: the communication frequency is the mean time between two communication phases; the communication size is the average size of application messages; the burstiness indicates whether all the application processes send data synchronously or create bursty traffic (for instance, collective operations create bursty traffic) and has an impact on the congestion. LU sends small messages asynchronously but very often. CG and MG communicate very often with medium message size in a burstiness way. BT and SP communicate quite often with medium message size but not in a burstiness way. Lastly, FT and IS do not communicate often but in a very burstiness way with big messages.

4.3.2 NPB completion time results

Figure 5 shows the NPB completion time results with MPICH2, MPICH2 with MPI5000 and MPICH-G2. All the results are relative to MPICH2 results.

FT and IS show very bad performances with our layer. As mentioned in the previous section, FT and IS use big size collective operations. Thus, each time a collective operation is called, the MPI5000 proxies become a bottleneck and the copy overhead is too important compared to the expected gain (see Section 3) of our architecture. To strengthen our idea, we run a simple `MPI_Alltoall` of 2 MB on 16 nodes without and with MPI5000. The completion time with MPI5000 is 2.74 higher than the one without MPI5000 which is similar to the result observed for FT on Figure 5 (2.86). The same conclusion could be done for IS, except that message sizes are smaller, and so is the overhead. The results obtained for LU and MG show that completion times with MPI5000 is a little higher than without MPI5000. We explain why in the next two sections. Finally, the BT, CG and SP results show that MPI5000 decreases their completion time and so is a valid solution to improve performances of some MPI applications executed on grids.

To conclude this section, firstly not all applications can benefit from MPI5000 but some of them really improve their execution time thanks to our proposition and secondly our MPI5000 proxies have to be optimized, which is not yet the case actually, in order to reduce their overhead.

4.3.3 MPI5000 benefits on idle timeouts

Table 4 shows the number of time the congestion window size is decreased without loss signal i.e., the number of idle timeouts we have measured with and without MPI5000 during the NPB execution. These figures are obtained thanks to the Web100 patch. 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. As expected, all NAS show a smaller number of idle timeouts with MPI5000.

In order to confirm these results, we disabled the slowstart after idle TCP feature (an option in linux). The results are shown in Table 5. CG, LU and MG show a similar completion time with or without the slowstart after idle. Thus, reducing the number of idle timeouts can not improve their performances. These results confirm our classification pre-

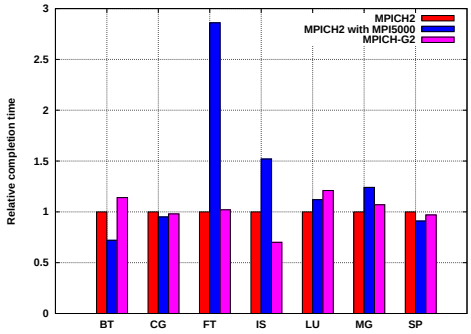

Figure 5. Relative completion time of NPB normalized to MPICH2.

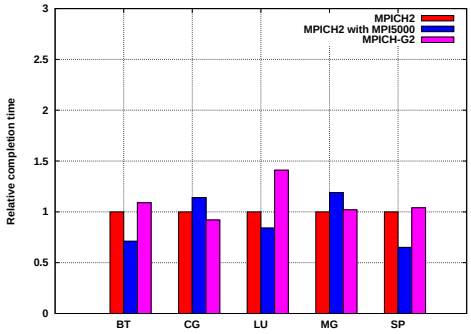

Figure 6. Relative completion time of NPB normalized to MPICH2 with cross-traffic.

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 without and with MPI5000.

NAS	MPICH2 without MPI5000		MPICH2 with MPI5000
	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.

sented in Table 3 because CG, MG and LU often communicate. For BT and SP, disabling slowstart after idle improves completion time. These results confirm that MPI5000 reduces the number of idle timeouts and then improves the execution time of benchmarks which suffer from them.

4.3.4 MPI5000 benefits 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 signal (neither duplicate ACK or RTO). Thus, the expected advantages of MPI5000 - that are to detect and repair losses faster, to reduce number of idle timeouts, to keep the congestion window closer to the available bandwidth - can not improve performances of CG, LU and MG in our previous experiments because neither idle timeouts nor losses appear.

In order to see what MPI5000 can improve under congestion conditions, we add cross-traffic, carried out with two `iperf` TCP flows at 1 Gbps on four extra nodes.

Figure 6 shows that BT, LU, and, SP benefit from MPI5000 in case of cross-traffic. However, CG decreases its performance compared to the case without cross traffic. This is probably due to a congestion window that do not increase fast enough but further investigation is needed on this point.

Table 6 shows the number of congestion signals without and with MPI5000 in case of cross-traffic. In the MPI5000's case, local refers to connections from one node to one proxy and distant refers to the connection between the proxies. As expected, the results show that MPI5000 deletes a lot of long distance RTO and DupACKs. However, despite of the cross-traffic, the number of losses measured on MG is not high enough to overlap the proxy overhead. But, its relative completion time is reduced compared to the case without cross-traffic. SP is also even better in this case than without cross-traffic and shows an improving completion time by 35% thanks to MPI5000.

4.3.5 Comparison with a Grid-enabled MPI implementation

Both Figure 5 and 6 show the MPICH-G2 performances with the NPB. They are very similar to MPICH2 performances except for IS due to grid-optimized collective operations of MPICH-G2. MPI5000 results outperform MPICH-G2 for BT, SP and LU in the same manner that it does for MPICH2. These results confirm that our approach

NAS	MPICH2 without MPI5000		MPICH2 with MPI5000			
	Distant		Local		Distant	
	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 without and with MPI5000 in case of cross-traffic.

is valid for MPI application executions on the grid and that MPI5000 is a complementary solution to grid-enabled MPI implementations. In future works, we should execute MPICH-G2 or GridMPI with MPI5000 to emphasize that MPICH-G2 or GridMPI on one side and MPI5000 on the other side do not address the same problems and are complementary.

5 Related works

Many MPI implementations are available for the grid like MPICH2 [5], OpenMPI [6], GridMPI [7] or MPICH-G2 [8]. Some of them need tuning to be efficiently executed with TCP in a grid [9]. Splitting TCP connections [10] has already been done in wireless or satellite networks but never in wire grid networks. We show in this paper that it is also useful for wired connections, mostly in the context of MPI application traffic features and with RTT values less than satellite's ones. Several past propositions [11] [12] [13] introduce proxies at the LAN/WAN interfaces to improve performance but the detailed propositions, the proxy level or usage are different from us. Even if we focalized on MPI applications, our proposition is not specific to MPI applications because it is at the socket level and thus, it would apply to any grid application executed above TCP.

6 Conclusion

The execution of MPI applications on grids faces new problems. We first show in this paper how MPI deals with TCP on grids. We conclude that (1) a high latency on the grid makes the application wait a longer time for DupACKs and for the TCP congestion window increase, (2) many RTO timeouts and idle times occur due to MPI application communication pattern. To take these problems into account, we propose MPI5000, a transparent layer that alleviates TCP's drawbacks by adding proxies at the LAN/WAN interfaces. It better manages the long-distance traffic and connections by splitting each LAN-WAN-LAN TCP connection in three distinct connections: LAN-LAN, WAN-WAN and LAN-LAN. Our proposition allows one to detect losses faster by avoiding RTO timeouts. MPI5000 also helps to avoid the slowstart phase after an idle time (time without communication on a link).

Finally, we evaluate MPI5000 on the french research grid, Grid'5000. Not all applications can benefit from MPI5000 but some of them really improve their execution time. Our evaluations confirm that MPI5000 effectively decreases the number of long-distance DupACKs and RTOs, the number of idle timeouts and thus the application execution time. The comparison with MPICH-G2 proves that MPI5000 can outperform a grid enabled MPI implementation. In conclusion, splitting TCP connections is a valid approach to better execute MPI applications on grids and MPI5000 can improve application performances.

In future works, we will reduce the MPI5000 proxy overhead by implementing a kernel version of the proxies to avoid two extra copies in user space. We should also study the scalability of proxy processes. Finally, we will implement and evaluate further optimizations now available thanks to MPI5000: bandwidth reservation between proxies, communication scheduling, parallel long-distance connections, use of an optimized/modified TCP protocol on the WAN.

Acknowledgment

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

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

[2] R. Bolze et al., "Grid'5000: a large scale and highly reconfigurable experimental Grid testbed." *International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, nov 2006, https://www.grid5000.fr/.

[3] A. J. Roy, I. Foster, W. Gropp, B. Toonen, N. Karonis, and V. Sander, "MPICH-GQ: quality-of-service for message passing programs," in *ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2000, p. 19.

[4] D. Bailey et al., "The NAS Parallel Benchmarks," NASA Ames Research Center, Tech. Rep. RNR-94-007, 1994.

[5] W. 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.

[6] E. Gabriel et al., "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation," in *Proceedings of 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, September 2004, pp. 97–104.

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

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

[9] L. Hablot, O. Glück, J.-C. Mignot, S. Genaud, and P. Vicat-Blanc Primet, "Comparison and tuning of MPI implementations in a grid context," in *IEEE conference on Cluster Computing*, September 2007, pp. 458–463.

[10] S. Kopparty, S. Krishnamurthy, M. Faloutsos, and S. Tripathi, "Split TCP for mobile ad hoc networks," *Global Telecommunications Conference*, vol. 1, pp. 138–142, Nov. 2002.

[11] R. Takano et al., "High Performance Relay Mechanism for MPI Communication Libraries Run on Multiple Private IP Address Clusters," in *Cluster Computing and the Grid (CCGrid'08)*, IEEE, Ed., Lyon, France, May 2008, pp. 401–408.

[12] M. Poeppe, S. Schuch, and T. Bemmerl, "A Message Passing Interface Library for Inhomogeneous Coupled Clusters," in *IPDPS*. IEEE, 2003.

[13] E. Gabriel, M. Resch, and R. Rhle, "Implementing MPI with optimized algorithms for metacomputing," in *Proc. of the Third MPI Developer's and User's Conference (MPIDC'99)*, 1999, pp. 31–41.