Grid Services for MPI

Camille Coti† Thomas Herault‡ Sylvain Peyronnet‡
coti@lri.fr *herault@lri.fr* *syp@lri.fr*
Ala Rezmerita‡ Franck Cappello†
rezmerit@lri.fr *fci@lri.fr*
†*INRIA, F-91893 Orsay France*
‡*Univ Paris Sud; LRI; INRIA; F-91405 Orsay France*

Abstract

Institutional grids consist of the aggregation of clusters belonging to different administrative domains to build a single parallel machine. To run an MPI application over an institutional grid, one has to address many challenges. One of the first problems to solve is the connectivity of the different nodes not belonging to the same administrative domain. Techniques based on communication relays, dynamic port opening, among others. have been proposed. In this work, we propose a set of Grid or Web Services to abstract this connectivity service, and we evaluate the performances of this new level of communication for establishing the connectivity of an MPI application over an experimental grid.

1 Introduction

In order to build very large systems, some institutions are willing to share their computing resources. This leads to parallel machines consisting of multiple clusters belonging to different administrative domains. Still, end users and programmers would like to deal with these systems in ways that are natural for traditional parallel machines. One example of such a behavior is to try to run an MPI application on instiutional grids. This is a challenge since administrative domains are not supposed to communicate easily. Security policies imply the installation of firewalls to protect the parallel application from outside attacks, like denial of service. This paper proposes connectivity methods for MPI processes running on nodes belonging to distinct administrative domains.

We present our implementation of the modified OpenMPI library and runtime environment originally adapted for the European project QosCosGrid. This implementation is called QCG-OMPI, standing for QosCosGrid-OpenMPI. The QosCosGrid project is concerned with grid environment for Quasi-Opportunistic Complex Systems Simulation. This project is described in [13]. [3] describes the QosCosGrid framework from its usecases' point of view. In order to provide this framework, several changes to the OpenMPI library and runtime environment have to be done. Developers will describe their communication patterns in order to let the system provide adequate resources for achieving performant use of the institutional grid; traditional grid tools will be used to provide access to the resources with single signon and certificate based authentication.

The first goal to achieve, and which is described in this work, is to grid-enable the OpenMPI runtime environment and library. Even if OpenMPI provides many useful features for running over a Grid such as heterogeneous CPUs compatibility, and multiple network interfaces handling chosen at runtime, it is still a challenge to port it over an institutional grid, mainly because of firewalls and complex network configurations.

The main contributions of this work are: A runtime environment to deploy an OpenMPI application over an institutional grid; Implementation and evaluation of Inter clusters communication and brokering services (ICCBS) for communications of the runtime environment and the library; and Integration of this ICCBS into the OpenMPI library and runtime environment.

The structure of the paper is as follows. We first present works related to our (section 2). Then we give in section 3 some insights about the architecture of our implementation. Mainly we describe the set of services we designed and how we integrated it within the OpenMPI library and the runtime environment. Section 4 is dedicated to the presentation of connectivity methods that allow nodes from distincts administrative domains to communicate. Last, in section 5, we present an experimental evaluation of our implementation and compare its performances to those obtained with OpenMPI without grid support and MPICH-G2.

[1] This work was partly supported by the EC grant for the QosCosGrid project (grant number: FP6-2005-IST-5 033883)

2 Related works

Several implementations of the MPI-2[9] standard are available, among them, one can distinguish two main open-source projects: MPICH[11] and OpenMPI[7]. The latter has been grid-enabled by the MPICH-G project [12] using the Globus toolkit [6] (GT). GT is a toolkit that aims to provide a set of software for an efficient use of grids. MPICH has been adapted to use the features of Globus in order to make an intensive use of the available resources for MPI applications. The launching process of an application uses Globus's features. The latest version is MPICH-G2, based on MPICH-1.2.7p1. MPICH implements the MPI-1.1 standard. MPICH-G2 uses Globus tools to allocate, spawn and manage jobs, and redirect standard input and outputs. Detection of the physical topology allows building topology-aware communicators. The general-purpose communicator can be split to create new communicators sharing the same network for a given depth. However, the application has to adapt its paralelization to the given topology at runtime.

MPICH-G2 can be used across several administrative domains, if firewalls are configured to have a specific range of open ports. Globus can be configured to use only those ports. But it does not include any firewall nor NAT bypassing, which is one of the features of our implementation. Another drawback of MPICH-G2 is its complex architecture. It uses an important number of external services, thus the startup time is an important phase of jobs' lifecycle; therefore MPICH-G2 is more performant on long applications.

PACX-MPI[8] is another grid-oriented MPI implementation. It was initially developed to interconnect two vendor parallel machines with their own vendor-based MPI implementation. It can also be used with clusters, allowing aggregating machines to form a meta-computer. It uses communication daemons that forward inter-cluster communications. It can be used across multiple administrative domains if the firewall allows at least one open port for the daemon communications. It can be a solution to bypass NAT if the daemons are located on the NAT server. But it does not provide any other firewall-bypassing feature. PACX-MPI can be configured for Globus compliance in order to use the GT mechanisms. It implements MPI-1.2 standard and some of MPI-2 standard.

Interoperable MPI (IMPI)[10] is a standard protocol for connecting multiple instances on MPI. It is used in the Japanese project GridMPI[1] [15], which is a grid-oriented implementation of the MPI-1.2 standard and most of MPI-2 features for high-performance MPI computing on multiple clusters. The latest public release of GridMPI is version 1.1. It does not support NAT, and requires global IP addressing. The IMPI standard limits the maximum number of MPI jobs to 32. Therefore, it is not adapted to large-scale computing. Moreover, GridMPI does not allow the use of heterogeneous environments because of different precision in floating point and 32/64 bits support, whereas OpenMPI allows heterogeneous 32/64 bits representation, and can handle more complex shiftings between different architectures.

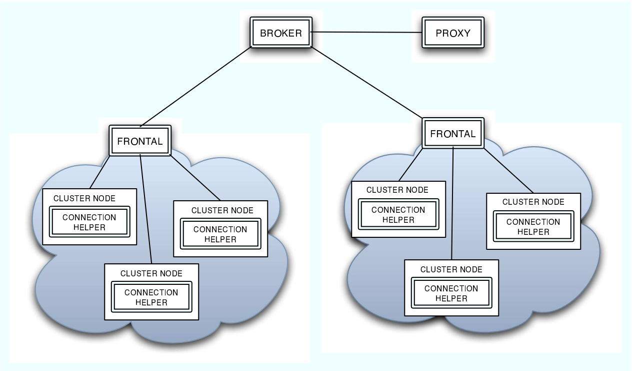

Figure 1. Inter-cluster communication services

3 Architecture

In this section, we first describe the set of services that were designed to help spawn and manage an MPI application spanning above multiple administrative domains, then we explain how these services have been integrated within the OpenMPI library and runtime.

Figure 1 presents a global picture of the architecture, across two administrative domains. Domains must be reachable one from the other, at least through gateway nodes, on which we run a first component, the Front-end component. In addition to services running on each administrative domain, we use two services that can be hosted by any administrative domain, namely the Broker and one or multiple proxies. The Broker service must be reachable from any Front-end component and the Proxy service must be reachable from any node. On all nodes of each administrative domain, we also use a connection helper service that is connected to the frontal service. The complete set of services builds a single connected component.

These services are provided as part of the infrastructure of the grid. This makes the weight of the runtime environment lighter, reducing the startup time of the application and improving its scalability.

All the services are implemented using the lightweight web-services engine gSOAP[4].

3.1 The Broker

The broker service is a centralized service running on only one machine in the grid and accepting incoming connections from frontal components.

The brokering service provides a way to communicate between nodes located in the same administrative domain and between nodes that would not be able to establish connections with each other otherwise, because of a NAT and/or a firewall.

To do this, the broker receives, from frontal service, the local cluster configuration. A cluster may be open on a port range, or authorize firewall bypassing techniques, or completely closed (in which case the only way to communicate with the cluster nodes is through a relaying technique).

In addition of collecting all the local configurations, the brokering service centralizes all processes contact information. This process contact information corresponds to the OpenMPI process unique identifier called process name (presented in Section 3.5) and the process access point (the public IP address of the node where the process is executed and the port number on which the process listens for incoming connections). The broker's global contact list is filled by every OpenMPI process and takes place every time an OpenMPI process creates an access point.

Each time the OpenMPI process wants to establish a communication with another OpenMPI process, it invokes the brokering service through connection-helper and frontal services. Cross checking all local configurations, the broker finds what technique is best fitted to establish the requested connection.

If a direct connection is possible between the OpenMPI processes, the brokering service returns the appropriate contact information to the initiator of the connection. Otherwise, the broker informs the frontal and connection-helper components that they must help OpemMPI process to apply the appropriate relaying technique. If both direct connection and relaying connection are possible, the direct connection is privileged.

3.2 The Front-end

The frontal service is running on a front-end machine of each cluster. It is connected to the brokering service and it accepts connections from connection-helper components.

During the update of nodes contact list and the initiation of connection establishment between two OpenMPI processes, the frontal service relays messages between the proxy service and the connection-helper. This hierarchical communication pattern provides scalability to the architecture, with a small cost on latency to connect to the Broker.

3.3 Connection-helper

The connection-helper service runs on every node. The goal of this component is to help OpenMPI process to establish the connection with another process by relaying messages between OpenMPI process and the frontal service before the connection establishment.

This component also plays very important role in the connection establishment using proxy and traversing-tcp techniques that will be described in more detail in the next section.

3.4 The Proxy

We assume that every QCG-OMPI process and every cluster node can access this service directly.

The proxy service is a service running on one or many machines in the grid and accepting incoming requests from the broker service.The goal of this service is to relay messages between nodes that are not able to establish a direct connection with each other.

Multiple proxy processes can be launched independently on the grid (as long as they are accessible from any point in the grid). The brokering service balances the load between all the available proxy services using a simple round-robin heuristic.

3.5 OpenMPI

OpenMPI startup is initiated by a first process (called *seed* daemon), which spawns processes on all the machines that will be used for the computation. Those processes are daemonized and keep running during all the execution, and beyond if requested. This set of daemons builds the runtime environment of OpenMPI. It supports the execution of the MPI application. Once all the daemons are initialized and ready, they locally spawn the MPI processes.

We support the execution of applications through a set of services which form an extension of the runtime environment that is part of the grid infrastructure. The purpose of this set of services is to deal with the issues raised by MPI computing on grids. The MPI library is interfaced with the services exhibited by the grid infrastructure in order to take advantage of it. Therefore the library becomes a client of those services.

Moreover, the runtime environment of the application also needs to communicate. Some communications might need some help from the services. For example, each local daemon opens a connection with the seed process during the initialization phase. Cross-domain computation require runtime-level cross-domain communications, and then need to use the brokering service. Thus, the local daemons are clients of the grid services too.

The initialization procedure of OpenMPI includes a phase in which processes exchange with each other some information through a campaign of many-to-one exchanges. Each OpenMPI runtime daemon, and each OpenMPI process sends its contact information to a central repository, which then broadcasts the whole contact informations to all. On large-scale systems and grids, those informations are provided on demand, only when a process needs them, for scalability reasons.

The broker knows the common communication media between two processes, and can provide them with the most efficient way to communicate with each other

4 Connectivity

The main issue we address here is how to let processes belonging to different administrative domains communicate efficiently. Administrative domains (AD) have firewall policies to protect themselves from the outside, including the others AD of the same grid. When two processes belonging to two different AD need to communicate one with the other, the AD have to change their firewall policy, or firewall bypassing techniques must be used.

Traditionaly, firewalls are configured so that outbound connections are authorized and inbound connections are blocked except for some specific ports such as the SSh port.

Most of the time, connectivity constraints limit the execution of parallel application on multiple sites. Connectivity problems can sometimes be solved when only one site uses a firewall: in this case, all the required connections are initiated from the firewalled site. However this solution implies modifications of applications or communication libraries changes. Moreover, if all sites are using firewalls this approach can no longer be applied. Another solution that can be used in order to solve connectivity problems is to configure the firewalls in order to open a port range and adapt the applications to use only theses port. However this solution is a threat to the site security.

Another problem that is encoutered is the so-called IPv4 address space problem. Network Address Translation (NAT) was introduced as a short term solution for solving it. Indeed, the solution is a new Internet protocol with a larger address space, namely IPv6. However, IPv6 deployment is progressing very slowly, so NATs still provide a valuable service.

In the rest of the section, we first describe basic techniques that address connectivity problems, then we present complex methods that allow connection through firewalls, thus solving the connectivity problems without threatening the security of the network.

4.1 Basic techniques

In the following we present some techniques that provide a framework for solving connectivity problems.

4.1.1 Direct Connection

The brokering service provides a way to communicate between nodes, storing all local configurations and all nodes access point. A contact list is filled when the OpenMPI process creates a new access point, i.e. OpenMPI process listens on a new port for incoming connections.

When the OpenMPI process on node A is willing to communicate with the OpenMPI process on node B, it invokes the brokering service through connection-helper and frontal services. As result of this new connection request, if node A is able to contact directly node B, the brokering service returns node B's contact point to node A.

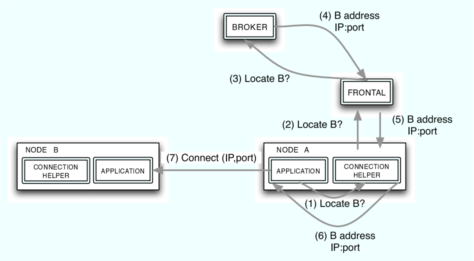

Figure 2. Connection between nodes belonging to the same cluster

Figure 4.1.1 shows the case when the two processors nodes A and B are located in the same cluster (there is no firewall connectivity restriction between them). Node A invokes the brokering service, asking to locate node B. The brokering service returns B's access point (IP address and port) allowing node A to open a direct connection to node B. When the frontal service receives from the broker service B's access point, it caches this information before forwarding it to the connection-helper.

Using this caching scheme, a process that issues the same request later will find this information closer.

If direct connection between two nodes is not possible (e.g. they are located in different administrative domains and there is a firewall connectivity restriction), the brokering service informs the frontal and the connection-helper about this restriction, meaning that the initiator of the connection must be helped in order to establish connection with distant node using relaying or Traversing TCP techniques, presented below.

4.1.2 Port Range technique

In case an open port range for incoming connections is available on the firewall of the grid site, the runtime environment ensures that all sockets used for incoming connections in this site are bound to a port in this range, thus guaranteeing that all the nodes located inside this site can be directly contacted from other sites. The OpenMPI process is informed about the known port range through Modular Component Architecture (MCA) parameters.

4.1.3 Relaying

The most reliable connectivity method implemented in QCG-OMPI is relaying. This technique is designed for the nodes that reside in different administrative domains, and that are such that their respective Firewall/NATs prevent either node from directly initiating a connection to the other. Relaying always works as long as both cluster nodes can connect to the proxy server. Instead of attempting a direct connection, the two nodes can simply use the proxy server to relay messages between them.

Its main drawbacks reside in its consumption of the proxy server's processing power and network bandwidth, and the induced communication latency between the nodes. Nevertheless, since there is no technique more efficient that works reliably on all existing firewall/NATs, relaying is a useful strategy.

When a node A needs to communicate with a node B in a remote cluster, it contacts its local connection-helper and requests it to provide a way to communicate with node B . Connection-helper forwards the connection demand to its frontal component. The frontal component then forwards the request to the broker component.

Once the new connection request is received, the broker asks the proxy service to create a new unique identifier (UID) for this new connection. Once the UID received, the broker notifies node A's frontal and node B's frontal. Both frontals will forward the message to the corresponding connection-helper process. Node A's and B's connection-helpers will give the MPI library the proxy address and the UID of the connection. Finally, node A and node B establish a direct connection with the proxy service that will perform the matchmaking between the nodes using the connection UID. To communicate with node B, node A sends the message to proxy server along its already-established client/server connection, and the proxy server forwards the message on to node B using its existing client/server connection with B.

4.2 Firewall traversing techniques

To enable communications between nodes located in different administrative domains, other techniques have been integrated in the service to increase the efficiency of the targeted grid, without reducing its security policy. We describe here the technique Traversing-TCP that have been proposed recently in [14].

4.2.1 Traversing TCP

Traversing TCP is derived from the TCP protocol and it works with firewalls that are not running stateful packet inspection. It essentially consists in transporting, using the broker, the initiating TCP packet (SYN) blocked by the firewalls or NAT on the server side and injecting the packet in the server IP stack. This technique is presented in [14].

5 Experimental results

In this section we present the performance measurements of our implementation. We conducted the experiments on two classical platforms of high performance computing: clusters of workstations with GigaEthernet network and computational grids. These experiments were done on the experimental Grid5000 platform or some of its components.

5.1 Experimental Platform

Grid5000 [5] is a physical platform featuring 13 clusters, each with 58 to 342 PCs, connected by the Renater French Education and Research Network. Grid5000 is a computer science project dedicated to the study of grids, and founded by the French government through the ACI Grid incentive action.

throughput in Mb/s	QCG-OMPI direct	QCG-OMPI proxy	QCG-OMPI traversing
Intra-Cluster	894.39	894.37	894.36
Orsay-Rennes	118.48		
Orsay-Bordeaux	136.08	76.40	138.18
Bordeaux-Rennes	131.68		

Table 1. Bandwidth comparison for all QCG-OMPI techniques (in Mb/s)

At the time we write this article, it gathers 1571 computers featuring four architectures (Itanium, Xeon, G5 and Opteron) distributed into 13 clusters over 9 cities in France.

For the two families of measurement we conducted (cluster and grid), we used only homogeneous clusters with AMD Opteron 248 (2.2 GHz/1MB L2 cache) bi-processors running at 2GHz. This includes 3 of the 13 clusters of Grid5000: the 93-nodes cluster at Bordeaux, the 312-nodes cluster at Orsay, a 99-nodes cluster at Rennes. Moreover, each node features 20GB of swap and SATA hard drive. Nodes are interconnected by a Gigabit Ethernet switch.

One of the major feature of the Grid5000 project is the ability for the user to boot all the computing nodes booked for her job into her own environment (including the operating system, distribution, libraries...). We used this feature to run all our measurements in an homogeneous environment. All the nodes were booted under linux 2.6.18.3. The tests and benchmarks are compiled with GCC-4.0.3 (with flag -O3). All tests are run in dedicated mode.

5.2 Latency and Bandwidth

First, we compare the latency and bandwidth of all connectivity techniques of our implementation with the two reference implementations: MPICH-G2 and OpenMPI without Grid features. To pursue this measurement, we used a simple ping-pong benchmark to compute the round time trip of messages of size 1 byte (for latency) and messages of size 10 Mbytes (for bandwidth). We used a grid composed of 3 clusters, each cluster holding 2 nodes. All values are summed up on tables 2 and 1. Values presented are mean values with a standard deviation less than 1%.

Table 2 presents the measured latencies. Values between any two pair of nodes were similar, so we present only this summed-up version. As expected, one can see that the intra-cluster communications are orders of magnitude lower than inter-cluster communications, for every implementation. MPICH-G2 presents a latency higher than the OpenMPI based implementations. This is due to the improvement of latencies in MPI implementations (MPICH and OpenMPI) since the version of MPICH on which MPICH-G2 is based.

For inter-cluster communications, direct and traversing techniques perform similarly to the MPICH-G2 implemen-

times in seconds	MPICH-G2	OpenMPI	QCG-OMPI direct	QCG-OMPI proxy	QCG-OMPI traversing
Intra-Cluster	0.0002	0.0001	0.0001	0.0001	0.0001
Orsay-Rennes	0.0104	0.0103	0.0103	0.0106	0.0103
Orsay-Bordeaux	0.0079	0.0078	0.0078	0.0084	0.0078
Bordeaux-Rennes	0.0081	0.0080	0.0080	0.0287	0.0080

Table 2. Latency comparison for all techniques (in seconds)

tation and the OpenMPI without grid support implementation. Since both MPICH-G2 and direct technique rely on opened ports, this is to be expected. For the traversing technique, one has to notice that the cost of connecting to the broker and inversing the connections are due at connection time only. Because the measurement begins only after the first connections is already established, this is transparent for the sustained latency achieved.

For the proxy technique, one can see that we measure a low overhead for inter-cluster communications between Orsay and Rennes and between Orsay and Bordeaux, but a significant overhead between Rennes and Bordeaux. This is due to the fact that in this experiment, we used a single relay for all communications, and this relay was located in the Orsay cluster.

Table 1 presents the measured bandwidths. As for latencies, we present only the summed-up version. Due to a hardware failure, we lost the Rennes cluster during our experiments, this is why the measurements including this cluster are not complete at the time we submit this paper. As expected, one can observe that the intra-clusters bandwidth is optimal, whatever technique is used. Inter-clusters bandwidths are optimal for direct and traversing techniques (since traversing technique only impact performances at startup time). The only technique that impact significantly the performances is the proxy technique, where the proxy network card is quickly saturated by all streams flowing through it.

5.3 Startup Time

Figure 3 presents the startup time of MPI applications for each of the compared implementations. The startup is measured as the time elapsed between the launching of the mpiexec command and the last exit of MPI_Init() calls. The connectivity technique used for the QCG-OMPI implementation was the direct technique. We used a single cluster (at Orsay) to evaluate the scalability of the startup. Because of connections between the runtime environment daemons, one can expect a latency and a startup time slightly slower with the proxy technique. However, the latencies measured in the previous experiment demonstrate that this would have a non-measurable impact on the performances of the startup time.

The MPICH-G2 implementation presents a startup time significantly slower and with a higher slope than the three other implementation. This is mainly due to the number of globus services which have to be started before the MPICH-G2 implementation can begin its work. Compared to our implementation MPICH-G2 provides more security functionalities through GSI, uses GRAM to startup each job, etc... This introduces a significant overhead in the startup time.

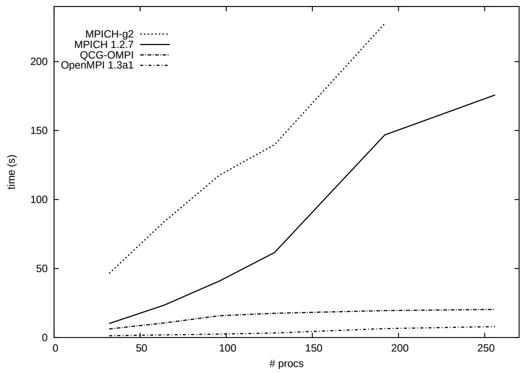

Figure 3. Startup time as function of the number of processors: QCG-OMPI vs. MPICH-G2 vs. OpenMPI

The startup phase of OpenMPI uses a set of all gather / broadcast to gather and distribute the contact information and setup the parallel application. When the number of nodes increases, the network becomes saturated with this contact informations and this introduces a bottleneck in the startup phase. With the QCG-OMPI implementation, this bottleneck appears sooner because the contact information is encapsulated for a significant part in web services, and thus occupies more space to be transfered on the network.

5.4 Benchmark Applications

There are currently no MPI application acknowledged by the community to benchmark the Grid. Most of the MPI applications assume an homogeneous network, and rely on this property to recover communications with computation. Although it is the goal of the QosCosGrid project to design MPI applications for the Grid, those applications are not ready to experiment with at this time. So, in order to validate our implementation and evaluate its overall performances on high-level applications, we use the traditional NAS benchmark suite [2], even if most of these benchmark do not scale well on heterogeneous platforms.

On each experiment, we distributed the number of used nodes (3/5 at Orsay, 1/5 at Rennes and 1/5 at Bordeaux). For all experiment using the proxy technique, we used a single proxy, located in the Orsay cluster. Ranks were assigned first to Orsay, then to Rennes and to Bordeaux. For example, an experiment with 25 nodes presents 15 nodes in Orsay (ranks 0 to 14), 5 nodes in Rennes (ranks 15 to 19) and 5 nodes in Bordeaux (ranks 20 to 24).

First, we evaluate the overhead of grid-enabling the MPI

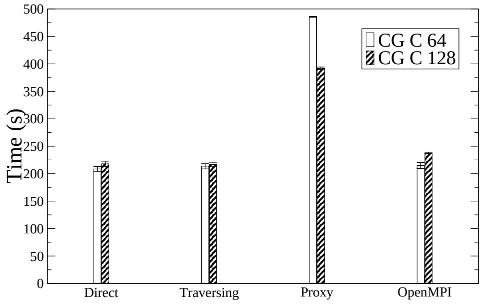

Figure 4. CG class C: execution time of QCG-OMPI and OpenMPI

library and runtime environment, comparing the three connectivity techniques to a reference time obtained with OpenMPI without grid support. To do this, we run the Conjugate Gradiant benchmark (CG) of class C with 64 and 128 nodes. The measurements presented in figure 4 are mean values measured for OpenMPI without grid support and QCG-OMPI with all techniques. Standard deviation is less than 4%.

CG is a latency-bounded benchmark: processes communicate using lots of small messages. Figure 4 shows that it doesn't scale well: excepted for proxy technique, execution takes longer with 128 processes than with 64 processes.

This slowdown is due to the fact that communications are no more overlapped by computation. Computation is shorter because of a finer-grain decomposition of the problem while communications suffer from a higher latency because of a network flood.

Using a proxy for inter-cluster communications introduces an extra hop, then increases the latency, and a bandwidth bottleneck. But this communication is not concerned by the intra-cluster network flood. Therefore, the inter-cluster latency added by the extra hop is overlapped by the latency due to the network flood, and the bandwidth bottleneck has a smaller impact because latency is the limiting factor of this application.

At smaller scale, inter-cluster communications through the proxy are not overlapped by intra-cluster communications nor computation and are slowing the execution down.

Now, we evaluate the scalability of our techniques, using the BT benchmark. We present in figure 5 the mean execution time of BT for different nodes size and different connectivity techniques.

As expected, the proxy method induces the highest overhead on the execution of an MPI application. The overhead caused by Traversing TCP is visible only when the connections are being established, so Traversing TCP and direct connection methods are equivalent once the connections are established. Since our benchmarks does not include connection establishment in the measurments, these two techniques show equivalent performances.

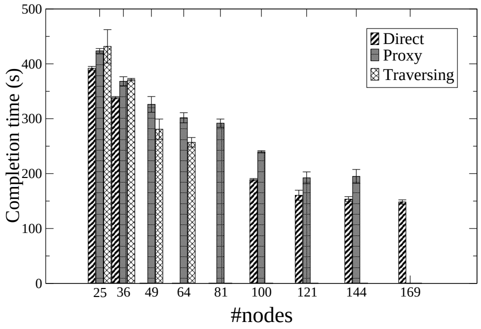

Figure 5. BT class C: mean execution time for the three QCG-OMPI connectivity techniques

A main characteristic of the BT benchmark is its high overlapping of communication by computation, so even running on a grid with high latency, the application still scales with the number of nodes. However, since messages are quite long, when using the proxy technique, the bottleneck of the proxy clearly impacts the performances.

6 Conclusion and future works

In this paper, we addressed the problem of grid-enabling the OpenMPI library for cluster of administrative domains. The main task we achieved is to provide complex and efficient inter-cluster connectivity techniques, which can span many different situations (ranging from the completely opened network to a network completely closed using firewalls and NATs).

We evaluated our implementation, and compared the efficiency of the different techniques to the state-of-the-art MPICH-G2 implementation; we also compared our performances with the base implementation of OpenMPI. These results demonstrated the fact that we can use with small overhead in latency and bandiwdth a cluster of clusters belonging to different administrative domains as a parallel machine.

Real experiments using BT and CG acknowledge this result. The new implementation has improved significantly the startup time as compared to MPICH-G2, by defining specific dedicated services to launch the application and provide connectivity techniques.

Within the QosCosGrid project, this implementation will be used as the base for the design and evaluation of applications fitted to the Grid.

Acknowledgements

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, and initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners.

Part of the authors are founded through the QosCosGrid European Project (grant number: FP6-2005-IST-5 033883).

References

[1] GridMPI: http://www.gridmpi.org.

[2] D. Bailey, T. Harris, W. Saphir, R. V. D. Wijngaart, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.0. Report NAS-95-020, Numerical Aerodynamic Simulation Facility, NASA Ames Research Center, 1995.

[3] M. Charlot, G. D. Fabritis, A. G. de Lomana, A. Gomez-Garrido, and D. G. et al. The QosCosGrid project: Quasi-opportunistic supercomputing for complex systems simulations. Description of a general framework from different types of applications. In *Ibergrid 2007 conference, Centro de Supercomputacion de Galicia (GESGA)*, 2007.

[4] R. A. V. Engelen and K. A. Gallivan. The gSOAP toolkit for web services and peer-to-peer computing networks. In *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, page 128, Washington, DC, USA, 2002. IEEE Computer Society.

[5] F. Cappello *et al.* Grid'5000: a large scale, reconfigurable, controlable and monitorable grid platform. In *proceedings of IEEE/ACM Grid'2005 workshop*, Seattle, USA, 2005.

[6] I. T. Foster. Globus toolkit version 4: Software for service-oriented systems. *J. Comput. Sci. Technol*, 21(4):513–520, 2006.

[7] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. 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.

[8] E. Gabriel, M. M. Resch, T. Beisel, and R. Keller. Distributed computing in a heterogeneous computing environment. In V. N. Alexandrov and J. Dongarra, editors, *PVM/MPI*, volume 1497 of *Lecture Notes in Computer Science*, pages 180–187. Springer, 1998.

[9] A. Geist, W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. L. Lusk, W. Saphir, A. Skjellum, and M. Snir. MPI-2: Extending the message-passing interface. In L. Bougé, P. Fraigniaud, A. Mignotte, and Y. Robert, editors, *Euro-Par, Vol. I*, volume 1123 of *Lecture Notes in Computer Science*, pages 128–135. Springer, 1996.

[10] W. L. George, J. G. Hagedorn, and J. E. Devaney. IMPI: Making MPI interoperable, Apr. 25 2000.

[11] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. High-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, September 1996.

[12] N. T. Karonis, B. R. Toonen, and I. T. Foster. MPICH-G2: A grid-enabled implementation of the message passing interface. *CoRR*, cs.DC/0206040, 2002.

[13] V. Kravtsov, D. Carmeli, A. Schuster, B. Yoshpa, M. Silberstein, and W. Dubitzky. Quasi-opportunistic supercomputing in grids, hot topic paper. In *IEEE International Symposium on High Performance Distributed Computing*, Monterey Bay California, USA, 2007.

[14] A. Rezmerita, T. Morlier, V. Néri, and F. Cappello. Private virtual cluster: Infrastructure and protocol for instant grids. In W. E. Nagel, W. V. Walter, and W. Lehner, editors, *Euro-Par*, volume 4128 of *Lecture Notes in Computer Science*, pages 393–404. Springer, 2006.

[15] R.Takano, M.Matsuda, T.Kudoh, Y.Kodama, F.Okazaki, and Y.Ishikawa. Effects of packet pacing for MPI programs in a grid environment. In *2007 IEEE International Conference on Cluster Computing (CLUSTER 2007) (9th CLUSTER'07)*. IEEE Computer Society, Dec. 2007.