



MPICH-GX (Grid-eXtension of MPICH)

Manual

Authors:

Oh-Young Kwon - oykwon@kut.ac.kr

Kum-Rye Park – namul@sogang.ac.kr

Kyung-Lang Park – lanx@parallel.yonsei.ac.kr

Oh-kyoung Kwon – okkwon@kisti.re.kr

Document Version: 0.9 Date: 2005-01-07



Copyright 2002-2005 Korea Institute of Scientist and Technology Information.
All rights reserved.

This document is licensed under the terms of the K*Grid Public License.
The details of K*Grid Public License is found at
<http://kmi.moredream.org/downloads/license.html>.

Contents

1	Introduction.....	4
1.1	What is MPICH-GX?	4
1.2	Architecture of MPICH-GX.....	4
1.2.1	File-based Initialization.....	5
1.2.2	Private IP Support and 'nproxy'.....	8
1.2.3	Efficient Collective Operations	9
1.3	Getting help	10
2	Installation and Configuration	11
2.1	Requirements	11
2.2	Installing required software	11
2.2.1	Installing Globus Toolkit.....	11
2.2.2	Extracting MPICH-1.2.6 source code	11
2.3	Installing MPICH-GX	11
2.3.1	Unzip MPICH-GX source files to \$GX_HOME	11
2.3.2	Installing patched MPICH-G2	12
2.3.3	Compile 'nproxy'	12
2.4	Environment Setup.....	12
2.4.1	Set environmental variables	12
2.4.2	Startup NAT Service on the front-end node variables.....	13
3	Running MPI applications.....	14
3.1	Compile your MPI program	14
3.2	Startup nproxy.....	14
3.3	Launching your MPI applications	14
3.3.1	Manual Launching.....	14
3.3.2	Launching by 'mpirun'	15
3.3.3	Launching by Job Submission Service (JSS) in GRASP	16
4	How to make 'process_info' files?	17
4.1	One process on one execution node	17
4.2	Two execution node	18
4.3	Four execution node.....	18
4.4	Four execution nodes in two clusters	18
5	Miscellaneous of this release and Future work	20
6	References	21

1 Introduction

This document is intended to provide the user with the information required to build, install, configure, manage, and use MPICH-GX.

1.1 What is MPICH-GX?

MPICH-GX is a patch of MPICH-G2 to extend functionality required in the Grid. MPICH-G2 is a well-defined implementation of Grid-enabled MPI, but it is needed to be modified for supporting some requirements of Grid applications. MPICH-GX provides following functions.

- 1) File-based Initialization
- 2) Private IP Support
- 3) Effective Collective Operations

Details of MPICH-GX are discussed in Section 1.2.

1.2 Architecture of MPICH-GX

If you are not familiar with the layered architecture of MPICH, it is better to skip this part for avoiding confusion.

Basically, MPICH-GX is based on MPICH-G2. Thus, it has similar architecture of MPICH-G2 as shown in Figure 1. It consists of a patch code of MPICH-G2 and a proxy daemon named 'nproxy'. The patch will modify 'globus2 device' of MPICH to add required functions. The nproxy will relay the messages of MPI processes when the process is located in private IP clusters. As shown in the Figure 1, application developer can make MPICH-GX applications by using standard MPI API. It can be mapped to the globus2 device passing through ADI layer. We modify the globus2 device, so that some functions in MPICH-GX are different from MPICH-G2.

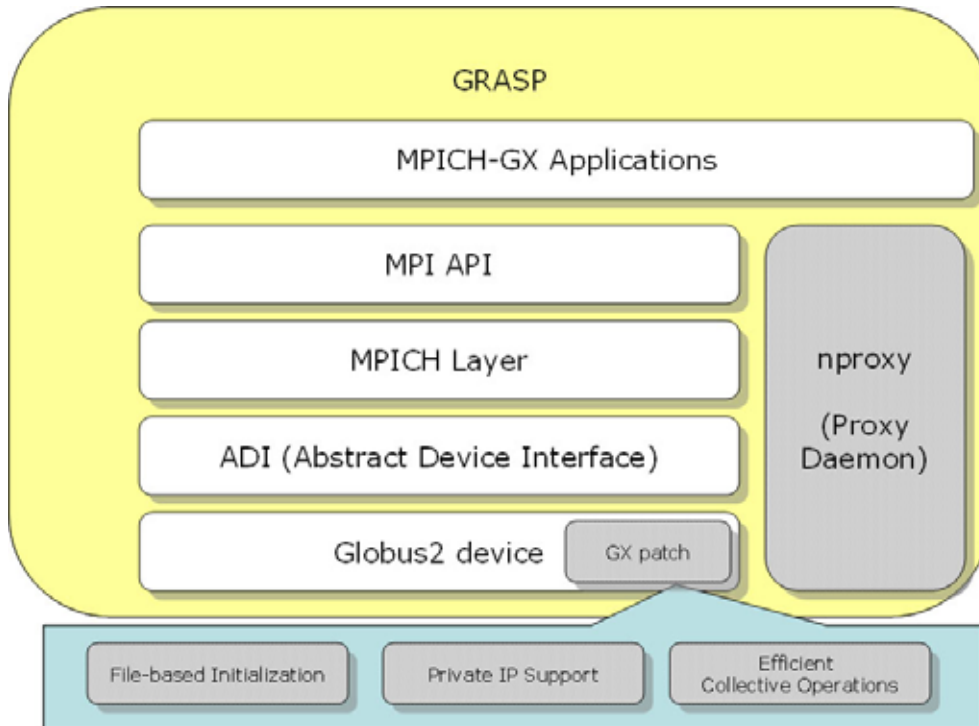


Figure 1. Architecture of MPICH-GX

1.2.1 File-based Initialization

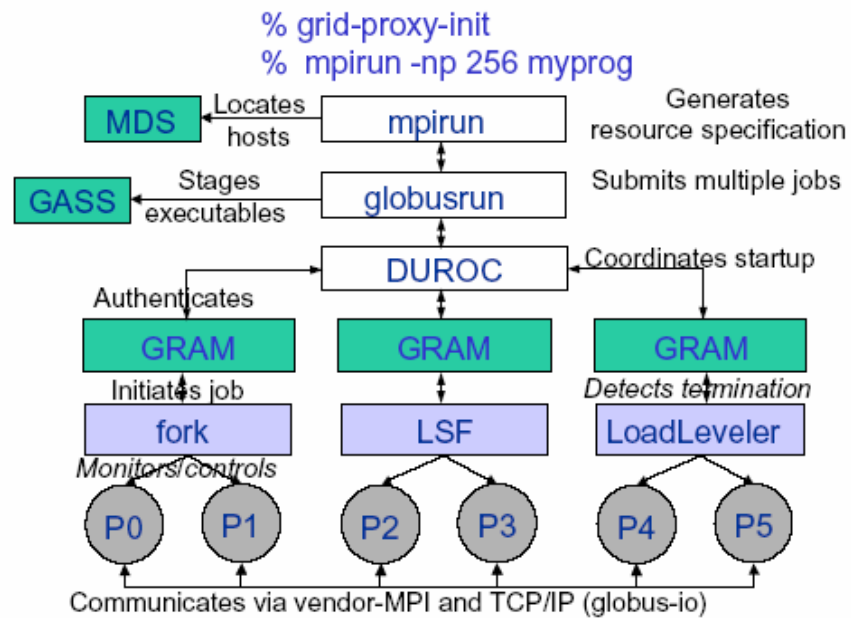


Figure 2. Procedure of the MPICH-G2 Startup

MPI applications need an initialization procedure that create processes and prepares necessary information for passing messages each other. Such procedure depends on the functions given by middlewares. For example, as illustrated in Figure 2, MPICH-G2 uses DUROC which is a component of Globus toolkit to initialize MPI processes. Sometimes it can be very efficient, but it makes the MPI library to be tied up by the middleware. Thus, the change of Globus toolkit results in inevitable change of MPI library. This is the reason why the MPICH-G2 cannot work with Globus toolkit 3.0 or later.

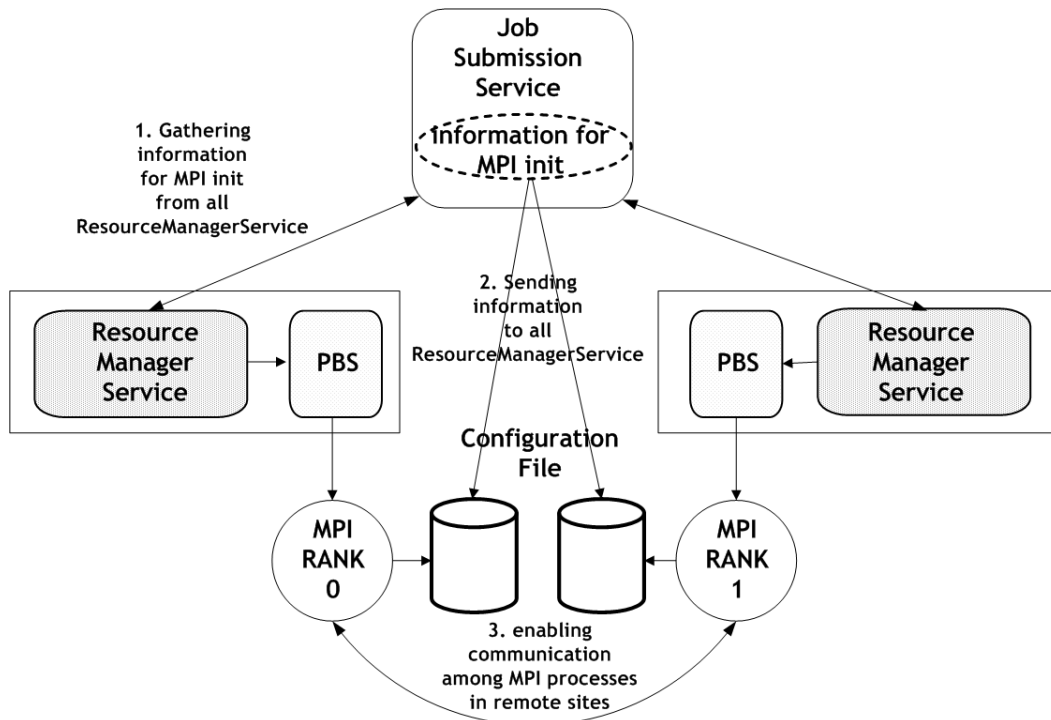


Figure 3. Process of File-based MPI Initialization in GRASP

Therefore, we modify the initialization procedure of MPICH-G2 which depends on DUROC component to file-based initialization. It allows users to launch MPI applications without DUROC. To use file-based initialization, users have to make the file (we call it configuration file). As showed in Figure 3, Job Submission Service (JSS) in GRASP helps you to run your MPI applications in convenient way. It can create a configuration file and stage the file to each execution node automatically. However, you also can launch the applications manually without any launching program if you know the method to make the configuration file. The contribution of the file-based initialization is that it is independent of middlewares. Even though the Grid middleware is changed more and more, we can launch the MPI application in a same way when we use the file-based initialization.

MPI applications need an initialization procedure that create processes and prepares necessary information for passing messages each other. Such procedure depends on the functions given

by middlewares. For example, MPICH-G2 use DUROC, which is a component of Globus, to initialize MPI processes. Sometimes it can be very efficient, but it makes the MPI library to be tied up by the middleware. Thus, the change of Globus results in inevitable change of MPI library. This is the reason why the MPICH-G2 cannot work with Globus 3.0 or later. Therefore, we modify the initialization procedure of MPICH-G2 which depends on DUROC component to file-based initialization. It allows users to launch MPI applications without DUROC. To use file-based initialization, users have to make the file (we call it configuration file). JobSubmission Service in GRASP helps you to run your MPI applications in convenient way. It can create configuration files and stage the files to execution node automatically. But, you also can launch the applications manually without any launching program if you know the method to make the configuration file. The contribution of the file-based initialization is that it is independent of middlewares. Even though the Grid middleware is changed more and more, we can launch the MPI application in a same way when we use the file-based initialization.

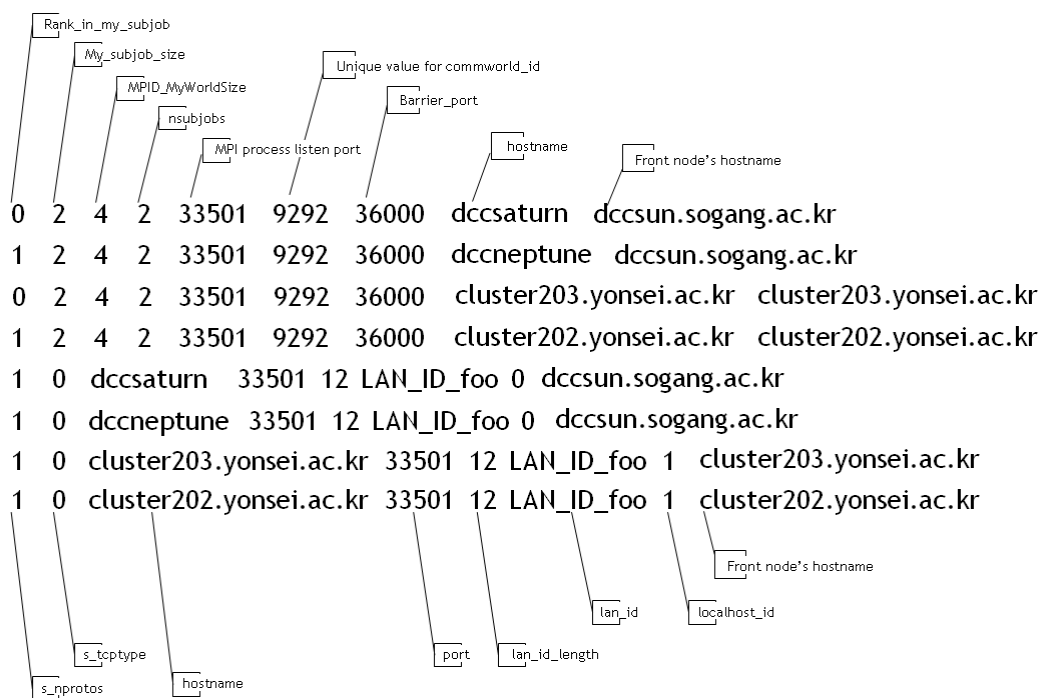


Figure 4. File Format for File-based MPI Initialization

We provide file format for file-based MPI initialization as depicted in Figure 4. File format composes of two parts: one is related to topology construction of MPI ranks and the other describes information to build channel based on topology. It could be made RANK sequentially through first topology part. Once it had construction for each RANK, initialization process could be established to build channel through second part.

First part is as following:

- **rank_in_my_subjob** means rank in one's subjob.
- **my_subjob_size** means size of one's subjob.
- **MPID_MyWorldSize** means total size of MPI job.
- **nsubjobs** means number of subjob.
- **MPI process listen port** means listening port of MPI process, which must be same value with the port of second part.
- **unique value for commworld_id** means unique id to construct COMMWORLD.
- **barrier_port** means listening port for synchronization between processes in COMMWORLD.
- **hostname** means hostname of computational node running each a MPI process.
- **front node's hostname** means hostname of front node connected with computational node running each a MPI process, in case with environment to use private IP addresses. Otherwise, it means hostname of computational node running each a MPI process. For instance, while first line of Figure 3 describes topology information which execution node is dccsaturn where front node is dccsun.sogang.ac.kr and has private IP address, third line of Figure 3 shows information which computational node is cluster203.yonsei.ac.kr where has public IP address..

Second part is as following:

- **s_nprotos** means kinds of used protocol.
- **s_tcptype** means type of used protocol: 0 is tcp, 1 is mpi, and 2 is unknown. Currently, tcp type could be supported.
- **hostname** means hostname of computational node running each a MPI process.
- **port** means listening port of MPI process.
- **lan_id_lng** means length of lan_id
- **lan_id** means identification that node is on the designated LAN.
- **localhost_id** means identification that node is on the designated intra-machine area rather than LAN or WAN.
- **front node's hostname** means hostname of front node connected with computational node running each a MPI process, in case with environment to use private IP addresses. Otherwise, it means hostname of computational node running each a MPI process.

1.2.2 Private IP Support and 'nproxy'

It is a well-known problem that MPICH-G2 does not support private IP clusters. In MPICH-G2,

processes communicate with each other based on IP addresses. Thus, it cannot support private IP clusters by the nature.

To support private IP clusters, MPICH-GX uses a communication relay scheme combining the NAT service with a user level proxy named 'nproxy.' In this approach, only incoming messages are handled by a user-level proxy to relay them into proper nodes inside the cluster, while the outgoing messages are handled by the NAT service at the front-end node of the cluster. Figure 5 show the conceptual diagram of our relay scheme.

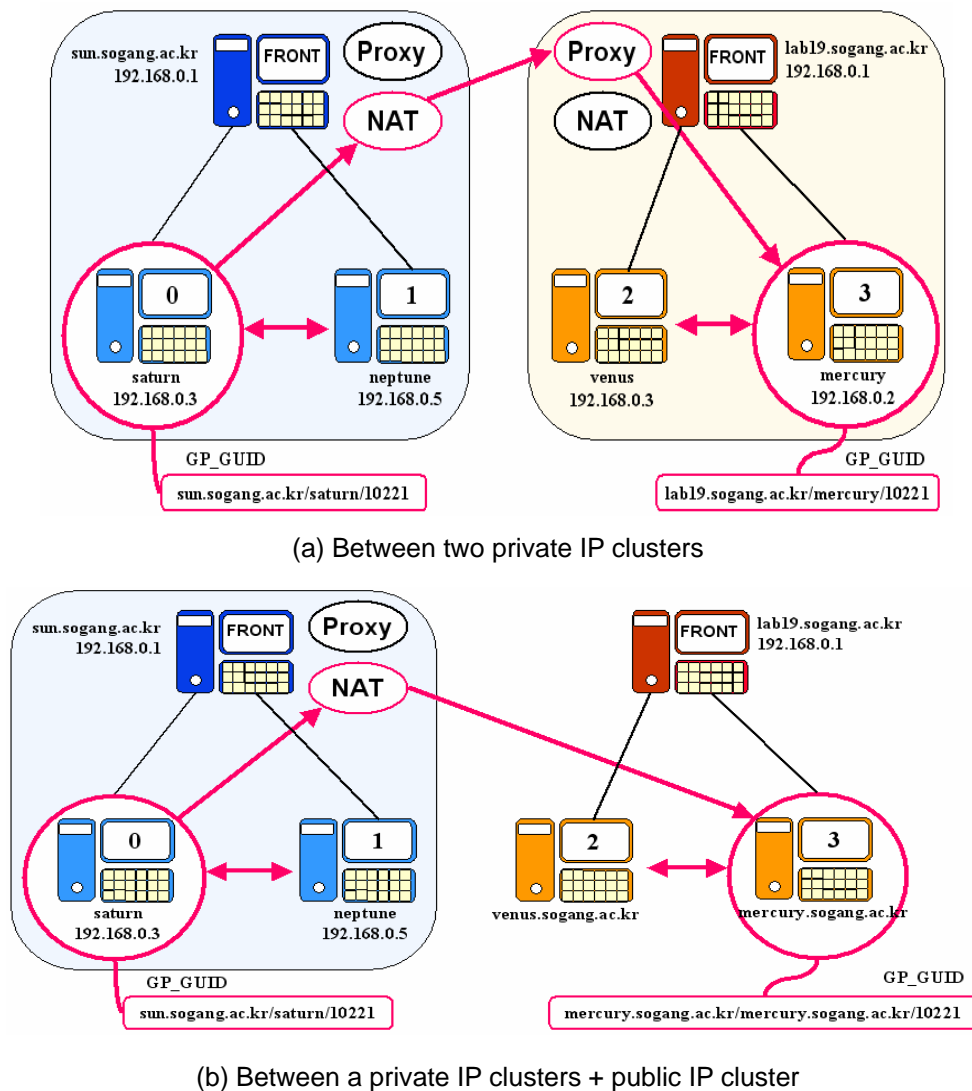


Figure 5. Communication Relay Scheme in MPICH-GX

1.2.3 Efficient Collective Operations

For the collective operations, MPICH-G2 uses the flat-tree algorithm which the root communicates with other processes directly (See details in <http://www.globus.org/mpi>.) But,

when if the high latency occurs between the root and other processes, its performance can be degraded significantly. Thus, MPICH-GX uses the HLOT (Hierarchical Latency Optimal Tree) algorithm for collective operations to avoid such high latency links. You can see details of HLOT algorithm in [1]

1.3 Getting help

If you have questions about GRASP or have found a bug, please send an email to oykwon@kut.ac.kr who is the leader of the MPICH-GX Project. For up-to-date information about MPICH-GX, please visit the web site <http://www.gridcenter.or.kr/kmi>.

2 Installation and Configuration

2.1 Requirements

- ✓ OS: Linux (RedHat 7.3 or more are recommended)
- ✓ Installing Globus Toolkit 2.4.3 or more
- ✓ Extracting MPICH 1.2.6 source code from a tar ball

2.2 Installing required software

2.2.1 Installing Globus Toolkit

You should install Globus toolkit 3.2.x before installing MPICH-GX. Download Globus Toolkit and see the install guide of Globus at <http://www.globus.org/>

Note: you must update packages related to Globus IO, XIO, and NEXUS via Globus Toolkit advisories at <http://www-unix.globus.org/toolkit/advisories.html>.

2.2.2 Extracting MPICH-1.2.6 source code

1. Download mpich-1.2.6 at <http://www-unix.mcs.anl.gov/mpi/mpich/download.html>

2. Unzip tar ball to \$MPICH_SRC

```
# tar xvfz mpich.tar.gz
```

You only need to extract source codes from the tar ball in this step. Full Installation of MPICH should be performed after applying MPICH-GX.

2.3 Installing MPICH-GX

Please make sure that all environmental variables are properly specified such as \$LD_LIBRARY_PATH, \$GLOBUS_PATH and \$GLOBUS_LOCATION.

2.3.1 Unzip MPICH-GX source files to \$GX_HOME

```
# tar xvfz mpich-gx.tar.gz
```

```
# cd mpich-gx
```

```
# cp gx_src/*.h $MPICH_SRC/mpid/globus2/
# cp gx_src/*.c $MPICH_SRC/mpid/globus2/
```

2.3.2 Installing patched MPICH-G2

This step is exactly same with the installation of MPICH-G2. (See details in MPICH-G2 homepage at <http://www.globus.org/mpi>.)

```
# $MPICH_SRC/configure --with-device=globus2:-flavor=gcc32 --prefix=$INST_DIR
# make
# make install
```

2.3.3 Compile 'nproxy'

We provide a makefile to compile the nproxy. You only need to type as followings.

```
# cd $GX_HOME/nproxy
# cat $MPICH_SRC/mpid/globus2/Makefile.mpich.header > globus_header
# make nproxy
```

2.4 Environment Setup

2.4.1 Set environmental variables

Add variables "FILENAME" and "FRONT_NAME" in your shell configuration script. FILENAME represent the name of configuration file for file-based initialization we mentioned in Section 1.2.1. FRONT_NAME stands for the FQDN of front-end node of the cluster. If you use public IP clusters, you need not to specify FRONT_NAME in your shell configuration.

```
# vi ~/.bashrc

export FILENAME={the name of process file}
export FRONT_NAME={FQDN of front-end node}
: wq

# source ~/.bashrc
```

2.4.2 Startup NAT Service on the front-end node variables

Our message relay scheme uses NAT service to send a message from a private IP node to others. You can startup Nat service by followings below commands. See NAT homepage to learn more about NAT services. ('\$' stands for the root shell.)

```
$ vi /etc/rc.d/rc.local
iptables -A POSTROUTING -t nat -o eth0 -j MASQURADE
echo 1 > /proc/sys/net/ipv4/ip_forward
:wq

$ /etc/init.d/xinetd restart
```

3 Running MPI applications

In this chapter, we describe the way to launch MPI applications by using MPICH-GX. Assume that a user hope to run a simple CPI program on 4 machines.

3.1 Compile your MPI program

You have to compile your MPI application by using MPICH-GX library. It is not different from the general compilation method of MPI programs.

```
# mpicc -o gx_cpi cpi.c
```

3.2 Startup nproxy

If you hope to use private IP clusters, you have to startup nproxy.

```
# $GX_HOME/nproxy/nproxy
```

3.3 Launching your MPI applications

Launching and initialization procedure of MPICH-GX is independent from the middleware, so that you can launch your application in various ways. You can use original 'mpirun' or 'globusrun' or you can even launch your application manually without any launching program.

3.3.1 Manual Launching

Manual launching must be a very inconvenient way, but it can help you understand entire steps of launching MPI applications. We will provide convenient launching program which can do followings automatically.

Making 'process_info' file

As stated above, you should make a configuration file to help MPI processes with performing the initialization procedure. Following is an example when a user runs 'cpi' application on 2 clusters each of which has 2 nodes. (Details of 'process_info' file is described in Section 4.)

```
# vi process_info

0 4 8 2 33501 9292 36000 cluster1 cluster101.yonsei.ac.kr
1 4 8 2 33501 9292 36000 cluster1 cluster102.yonsei.ac.kr
0 4 8 2 33501 9292 36000 dccsun dccsaturn.sognag.ac.kr
```

```

1 4 8 2 33501 9292 36000 dccsun dccneptune.sognag.ac.kr
1 0 cluster1 33501 6 WAN_ID1 1 cluster101.yonsei.ac.kr
1 0 cluster1 33501 6 WAN_ID1 1 cluster102.yonsei.ac.kr
1 0 dccsun 33501 6 WAN_ID0 0 dccsaturn.sogang.ac.kr
1 0 dccsun 33501 6 WAN_ID0 0 dccneptune.sogang.ac.kr

# cp process_info FILE_NAME

```

Staging 'process_info' file to all execution nodes

```

# echo $FILE_NAME
    /home/globus/process_info
# scp /home/globus/process_info user@execution_node:FILE_NAME

```

Staging executables

```

# scp /home/globus/cpi user@execution_node:/home/globus/cpi

```

Run executables on each remote machine

```

cluster101]# /home/globus/cpi RANK=0
cluster102]# /home/globus/cpi RANK=1
dccsaturn]# /home/globus/cpi RANK=2
dccneptune]# /home/globus/cpi RANK=3

```

3.3.2 Launching by 'mpirun'

You also can launch MPI application by using old 'mpirun.'

Making and Staging 'process_info' file.

It is same with above making and staging 'process_info' file

Making 'machine file'

```

# vi machinefile

"cluster1/jobmanager-pbs" 2
"dccsun/jobmanager-pbs" 2
:wq

```

Launching executable by using 'mpirun'

```
# mpirun -np 4 -machinefile machinefile -s cpi
```

3.3.3 Launching by Job Submission Service (JSS) in GRASP

You can even use launch your MPICH-GX applications by using GRASP. JSS in GRASP can provide a convenient way to run your applications. See details in [4].

In addition to mentioned methods, you can even make an own script to launch your applications if you know the method to write configuration files. You can learn about it in Section 4.

4 How to make 'process_info' files?

We will describe the way to make 'process_info' files. We will explain with CPI example.

4.1 One process on one execution node

```
# cat process_info1
0 1 1 0 33501 9292 33519 front.example.com node06.example.com
1 0 node06.example.com 33501 10 LAN_ID_foo 0 front.example.com
```

This file stands for followings:

- The first value in the first line '0': rank in my subjob
- The second value in the first line '1': size of my subjob
- The third value in the first line '1': the number of processes in the application
- The 4th value in the first line '0': the number of subjobs
- The 5th value in the first line '33501': MPI process listen port
- The 6th value in the first line '9292': unique value for commworld_id
- The 7th value in the first line '36000': MPI process barrier port
- The 8th value in the first line 'front.example.com':
- The first value in the second line '1': number of protocols (1 is required)
- The second value in the second line '0': type of the protocol (0 stands for TCP)
- The third value in the second line 'node06.example.com': hostname of execution node
- The 4th value in the second line '33501': MPI process listen port
- The 5th value in the second line '10': LAN_ID length
- The 6th value in the second line 'LAN_ID_foo': LAN_ID of the process
- The 7th value in the second line '0': subjob number
- The 8th value in the second line 'front.example.com':

```
# export FILENAME=./process_info1
# ./cpi
Process 0 on node06.example.com
Pi is approximately 3.1416009869231254, Error is 0.00000833333333323
Wall clock time = 0.000171
```

4.2 Two execution node

In this case, the number of subjob is 1 and size of subjob is 2. So, the process_info file should be written as follows:

```
# cat process_info2
0 2 2 1 33501 9292 33519 front.example.com node06.example.com
1 2 2 1 33501 9292 33519 front.example.com node07.example.com
1 0 node06.example.com 33501 10 LAN_ID_foo 0 front.example.com
1 0 node07.example.com 33501 10 LAN_ID_foo 0 front.example.com
```

4.3 Four execution node

In this case, the number of subjob is 1 and size of subjob is 4. So, the process_info file should be written as follows:

```
# cat process_info4
0 4 4 1 33501 9292 33519 front.example.com node06.example.com
1 4 4 1 33501 9292 33519 front.example.com node07.example.com
2 4 4 1 33501 9292 33519 front.example.com node08.example.com
3 4 4 1 33501 9292 33519 front.example.com node09.example.com
1 0 node06.example.com 33501 10 LAN_ID_foo 0 front.example.com
1 0 node07.example.com 33501 10 LAN_ID_foo 0 front.example.com
1 0 node08.example.com 33501 10 LAN_ID_foo 0 front.example.com
1 0 node09.example.com 33501 10 LAN_ID_foo 0 front.example.com
```

4.4 Four execution nodes in two clusters

In this case, the number of subjob is 2 and size of subjob is 2. So, the process_info file should be written as follows:

```
# cat process_info22
0 2 4 2 33501 9292 33519 front.example.com node06.example.com
1 2 4 2 33501 9292 33519 front.example.com node07.example.com
0 2 4 2 33501 9292 33519 front.example.com node08.example.com
```

```
1 2 4 2 33501 9292 33519 front.example.com node09.example.com
1 0 node06.example.com 33501 10 LAN_ID_foo 0 front.example.com
1 0 node07.example.com 33501 10 LAN_ID_foo 0 front.example.com
1 0 node08.example.com 33501 10 LAN_ID_foo 1 front.example.com
1 0 node09.example.com 33501 10 LAN_ID_foo 1 front.example.com
```

5 Miscellaneous of this release and Future work

Unfortunately, this release does not include all functions of MPICH-GX and support limited version of Globus Toolkit and MPICH. Followings are known problems and our future works. We will fix such problems and release stable version soon.

- This release does not include efficient collective operations algorithm.
- The patch in this release includes is only for the MPICH-1.2.6 release.
- This release does not include convenient launching tools.
- Format of the 'process_info' file is too difficult for users to write manually. We will simplify the format.

6 References

- [1] Kyung-Lang Park et al. "Design and Implementation of a Dynamic Communication MPI Library for the GRID," International Journal of Computers and Applications, Vol. 26, No. 3, 2004, pp. 165-172
- [2] Kum-Rye Park et al., "MPICH-GP: A Private-IP-enabled MPI over Grid Environments," In Proceeding of the 2nd International Symposium on Parallel and Distributed Processing and Applications (ISPA 2004).
- [3] Si-Youl Choi et al., "An NAT-Based Communication Relay Scheme for Private-IP-enabled MPI over Grid Environments," In Proceeding of the International Conference on Computational Science 2004 (ICCS 2004). June 2004. pp. 499-502.
- [4] Administrator's Guide of GRASP, online at <http://kmi.moredream.org>