mpi-start

MPI-START

MPI-START AND MPI-UTILS

Version:	**1.5.2**
Document version:	**1.0.6**
Date:	**July 30, 2013**

CONTENTS

1 ABOUT

1.1 MPI-START

1.1.1 DESCRIPTION

mpi-start is a set of scripts to close the gap between the workload management system of a Grid insfrastructure and the configuration of the nodes on which MPI applications are run. The package is used to help the user to start MPI applications on heterogeneous Grid sites.

mpi-start provides an abstraction layer that offers a unique interface to start parallel jobs with different execution environments implementations. It supports several different MPI implementations under different batch systems.

mpi-start was originally developed in the frame of the int.eu.grid project for the execution of MPI applications with the CrossBroker metascheduler and then extended its use as the official way of starting MPI jobs within EGEE. mpi-start was then included in the EMI project.

Development is tracked at http://github.com/IFCA/mpi-start.

1.1.2 REQUIREMENTS

mpi-start only requires bash compatible shell for working. mpi-start uses several commands that are available in most unix systems: readlink, mount and mktemp.

1.1.3 SOURCE CODE

Source code is available at mpi-start git repository (https://github.com/IFCA/mpi-start). Released versions are tagged as `mpi-start_R_X_Y_Z-r`, where X.Y.Z is the mpi-start version and r the revision number for that version.

1.2 MPI-UTILS

1.2.1 DESCRIPTION

MPI-utils is a metapackge (emi-mpi) that depends on mpi-start and a yaim plugin for easy configuration of the MPI support in grid sites.

1.2.2 REQUIREMENTS

MPI-Utils contains a yaim plugin, therefore it needs yaim-core installation available. The plugin does not use any non-standard tools.

1.2.3 SOURCE CODE

The source code of the yaim plugin is available at glite CVS (http://jra1mw.cvs.cern.ch/cgi-bin/jra1mw.cgi/org.glite.yaim.mpi) under org.glite.yaim.mpi module. Released versions are tagged with tags in the form `glite-yaim-mpi_R_X_Y_Z_r`, where X.Y.Z is the plugin version and r the revision number for that version.

2 USER GUIDE

2.1 INSTALLATION

Normally users do not need to install mpi-start. However if they want to use it in a site without an existing installation, the recommendation is to create a tarball installation that can be transfered in the input sandbox of the job.

In order to create a tarball installation, get the source code and do the following:

```
$ make tarball
```

This will create a mpi-start-X.Y.Z.tar.gz (with X.Y.Z being the version of mpi-start) that contains all that is needed for the execution of jobs. In your job script unpack the tarball and set the `I2G_MPI_START` environment variable to `$PWD/bin/mpi-start`.

2.2 USAGE

mpi-start can be controlled via environment variables or command line switches, most configuration dependent paramenters are automatically detected by mpi-start and do not need to be specified by the user. The following command line will be enough to run the application with the site defaults:

```
$ mpi-start application [application arguments ...]
```

2.3 COMMAND LINE OPTIONS

-h show help message and exit

-V show mpi-start version

-t mpi_type use `mpi_type` as MPI implementation

-v be verbose

-vv include debug information

-vvv include full trace

-pre hook use `hook` as pre-hook file

-post hook use `hook` as post-hook file

-pcmd cmd use `cmd` as pre-command

-npnode n start n processes per node

-pnode start only one process per node (equivalent to -npnode 1)

-npsocket n start n processes per CPU socket

-psocket start only one process per CPU socket (equivalent to -npsocket 1)

-npcore n start n processes per core

-pcore start only one process per core (equivalent to -npcore 1)

-np n set total number of processes

-i file use `file` as standard input file

-o file use `file` as standard output file

-e file use `file` as standard error file

-x VAR[=VALUE] define variable VAR with optional VALUE for the application's environment (will not be seen by mpi-start!)

-d VAR=VALUE define variable VAR with VALUE

– optional separator for application and arguments, after this, any arguments will be considered the application to run and its arguments

For example, the following command line would start /bin/hostname 3 times for available node using Open MPI:

```
$ mpi-start -t openmpi -npnode 3 -- /bin/hostname
```

2.4 ENVIRONMENT VARIABLES

Prior to version 1.0.0 mpi-start only used environment variables to control its behavior. This is still possible, although command line arguments will override the environment variables defined. Next table shows the complete list of variables, with the command line options that can be used to set them:

Variable	Cmd line	Meaning
I2G_MPI_APPLICATION		The application binary to execute.
I2G_MPI_APPLICATION_ARGS		The command line parameters for the application
I2G_MPI_TYPE	-t	The name of the MPI implementation to use.
I2G_MPI_PRE_RUN_HOOK	-pre	This variable can be set to a script which must define the pre_run_hook function. This function will be called after the MPI support has been established and before the internal pre-run hooks. This hook can be used to prepare input data or compile the program.
I2G_MPI_POST_RUN_HOOK	-post	This variable can be set to a script which must define the post_run_hook function. This function will be called after the mpirun has finished.
I2G_MPI_START_VERBOSE	-v	Set to 1 to turn on the additional output.
I2G_MPI_START_DEBUG	-vv	Set to 1 to enable debugging output
I2G_MPI_START_TRACE	-vvv	Set to 1 to trace every operation that is performed by mpi-start
I2G_MPI_APPLICATION_STDIN	-i	Standard input file to use.
I2G_MPI_APPLICATION_STDOUT	-o	Standard output file to use.
I2G_MPI_APPLICATION_STDERR	-e	Standard error file to use.
I2G_MPI_SINGLE_PROCESS	-pnode	Set it to 1 to start only one process per node.
I2G_MPI_PER_NODE	-npnode	Number of processes to start per node.
I2G_MPI_SINGLE_SOCKET	-psocket	Set it to 1 to start only one process per CPU socket.
I2G_MPI_PER_SOCKET	-npsocket	Number of processes to start per CPU socket.
I2G_MPI_SINGLE_CORE	-pcore	Set it to 1 to start only one process per core.
I2G_MPI_PER_CORE	-npcore	Number of processes to start per core.
I2G_MPI_NP	-np	Total number of processes to start.

These variables can also be set with the `-d` command line switch. The following example shows how to set the `I2G_MPI_TYPE` variable to `openmpi`:

```
mpi-start -d I2G_MPI_TYPE=openmpi
```

There are also other variables that can modify the behaviour of mpi-start, but they are described in other sections of this document. The ones dealing with site configuration of mpi-start are documented in the Site Administrator Section (4), and the variables dealing with the Hooks are summarized in Section 3.5.

2.5 SCHEDULER AND EXECUTION ENVIRONMENT SUPPORT

mpi-start support different combinations of batch schedulers and execution environments using plugins. The schedulers are automatically detected from the environment and the execution environment can be selected with the `I2G_MPI_TYPE` variable or the `-t` command line option.

Scheduler Plugins	
`sge`	supports Grid Engine.
`pbs`	for supporting PBS/Torque.
`lsf`	supports LSF.
`condor`	gives support for Condor. This plugin lacks the possibility to select how many processes per node should be run.
`slurm`	for supporting Slurm. As with condor, the plugin currently lacks the processes per node support.

Execution Environment Plugins	
`openmpi`	Open MPI
`mpich2`	MPICH2
`mpich`	MPICH
`lam`	LAM-MPI
`pacx`	PACX-MPI
`dummy`	Debugging environment, just executes application in current host.

3 HOOKS

The mpi-start Hooks Framework allow the extension of mpi-start features without changing the core functionality. Several hooks are included in the default distribution of mpi-start for dealing with file distribution and some MPI extensions. Site admins can check the local hooks description while users probably are interested in developing their own hooks for compilation.

3.1 FILE DISTRIBUTION HOOKS

File distribution hooks are responsible for providing a common set of files prior to the execution of the application in all the hosts involved in that execution. Three steps are taken for file distribution:

- Detection of shared filesystem: mpi-start will try to detect if the current working directory is in a network file system (currently considered as such are: nfs, gfs, afs, smb, gpfs and lustre). If the detection is positive, the distribution of files is not performed. Detection can be totally skipped by setting: `MPI_START_SHARED_FS` to 0 or 1 (0 means that mpi-start will try distribution in the next step, and 1 that it won't).
- File distribution: in this step, mpi-start copies files from the current host to the other hosts involved in the execution. It uses the most suitable of the available distribution methods. Distribution methods are plugins that are detected at runtime by checking all the files with `.filedist` extension in the mpi-start `etc` directory.
- Clean-up: once the job is finished, mpi-start will try to clean-up in the remote hosts the files that were copied previously using the same distribution method as in the previous step. **Clean-up removes entire directories, so it should not be used when running from important paths (e.g. home directory)!**. The clean-up step can be deactivated by setting `MPI_START_DISABLE_CLEANUP` to `yes`.

The file distribution method can be enforced by using the `I2G_MPI_FILE_DIST` variable.

3.1.1 DISTRIBUTION METHOD PLUGINS

A file distribution plugin must contain the following functions:

- `check_distribution_method()`: called during initialization to check if the distribution method is suitable. It returns a number, the lower the number it returns, the higher priority it will have. If the method is not suitable, then it should return 255 or larger.
- `copy()` perform the actual copy of files between hosts. Files are in a gzipped tarball pointed by the `TARBALL` variable.
- `clean()`: clean up any files once the execution is finished.

These distribution methods are included in mpi-start:

ssh uses scp to copy files, needs passwordless ssh properly configured.

mpiexec uses OSC mpiexec for copying, needs OSC mpiexec installed.

cptoshared_area copies files to a shared area that is not the current working directory. Needs the following variables:

- `MPI_SHARED_HOME`: set to "`yes`".
- `MPI_SHARED_HOME_PATH`: path of the shared area that will be used for execution.

mpi_mt uses mpi_mt for copying the files. Needs the mpi_mt binary to be available in all machines.

3.2 EXTENSIONS HOOKS

Extension hooks are local site hooks that come in the default mpi-start distribution. The following hooks are available:

Affinity The Affinity hook is enabled by setting the `MPI_USE_AFFINITY` variable to `1`. When enabled (and the execution environment supports it), it will define the appropriate options for setting the processor affinity under the selected MPI implementation.

OpenMP The OpenMP hook is enabled by setting the `MPI_USE_OMP` variable to `1`. When enabled it will define the `OMP_NUM_THREADS` environment variable to the number of processors available per mpi process.

MPItrace MPItrace is enabled by setting the `I2G_USE_MPITRACE` variable to `1`. It adds to the execution the mpitrace utility, assuming it is installed at `MPITRACE_INSTALLATION`. Once the execution is finished, it gathers and creates the output files at the first host.

MARMOT Marmot is a tool for analysing and checking MPI programs. This hook enables the use of the tool if the variable `I2G_USE_MARMOT` is set to `1`. It also copies the analysis output to the first host.

Compiler This hook sets environment variables `MPI_MPI<COMPILER>`, where `COMPILER` is one of `CC`, `F90`, `F77`, `CXX`, for C, FORTRAN 90, FORTRAN 77 and C++ compilers respectively. This variables should point to valid compilers for the current MPI implementation. The hook also fixes compiler flags (`MPI_MPIxx_OPTS`) to avoid problems with bad flag for the current processor architecture. This hook can be disabled by setting the environment variable `MPI_COMPILER_HOOK` to `0`.

These hooks can be completely removed by deleting the affinity.hook, openmp.hook, mpitrace.hook, marmot.hook, or compiler.hook in the mpi-start configuration directory.

3.3 LOCAL SITE HOOKS

Site admins can define their own hooks by:

- Creating new `.hook` files in the configuration directory, or
- modifying the `mpi-start.hooks.local` file.

The `.hook` files are executed in alphabetical order and the `mpi-start.hooks.local` will be executed after any other hook in the system are executed and the shared file system detection is performed. Each hook file contains the following functions:

- `pre_run_hook ()`: it will be executed before the user hooks and the user application gets executed.
- `post_run_hook ()`: it will be executed after the user application gets executed.

If any of these functions is not available, the hook will be ignored.

3.4 DEVELOPING USER HOOKS

Users can also customize the mpi-start behavior defining their own hooks by using the `-pre` and `-post` command line switches or by setting the `I2G_MPI_PRE_RUN_HOOK` and `I2G_MPI_POST_RUN_HOOK` environment variables

`-pre` / `I2G_MPI_PRE_RUN_HOOK` path of the file containing the pre-hook, in this file a function called `pre_run_hook()` must be available. This function will be called before the application execution. The pre-hook can be used, for example, to compile the executable itself or download data.

`-post` / `I2G_MPI_POST_RUN_HOOK` path of the file containing the post-hook, in this file a function called `post_run_hook()` must be available. This function will be called once the applications finishes its execution. The post-hook can be used to analyze results or to save the results on the grid.

Both pre and post hooks can be in the same file. Next sections contain some hook examples

3.4.1 COMPILATION

Pre-run hook can be used for generating the binaries of the application that will be run by mpi-start. The following sample shows a hook that compiles an application using the C MPI compiler, as defined by the compiler hook in the `MPI_MPICC` variable. It assumes that the source code is called like the application binary, but with a `.c` extension. Use of complex compilation commands like configure, make, etc is also possible. This code is only executed in the first host. The results of the compilation will be available to all hosts thanks to the file distribution mechanisms.

```
#!/bin/sh

# This function will be called before the execution of MPI application
pre_run_hook () {

  # Compile the program.
  echo "Compiling ${I2G_MPI_APPLICATION}"
  $MPI_MPICC $MPI_MPICC_OPTS -o ${I2G_MPI_APPLICATION} ${I2G_MPI_APPLICATION}.c
  if [ ! $? -eq 0 ]; then
    echo "Error compiling program.  Exiting..."
    return 1
  fi
  echo "Successfully compiled ${I2G_MPI_APPLICATION}"
  return 0
}
```

3.4.2 INPUT PREPROCESSING

Some applications require some input preprocessing before the application gets executed. For example, gromacs has a `grompp` tool that prepares the input for the actual `mdrun` application. In the following example the `grompp` tool prepares the input for gromacs:

```
#!/bin/sh

pre_run_hook()
{
   echo "pre_run_hook called"

   # Here comes the pre-mpirun actions of gromacs
   export PATH=$PATH:/$VO_COMPCHEM_SW_DIR/gromacs-3.3/bin
   grompp -v -f full -o full -c after_pr -p speptide -np $MPI_START_NP

   return 0
}
```

Note the use of the `MPI_START_NP` variable to get the number of processors. See the developer section for a list of internal mpi-start variables.

3.4.3 OUTPUT GATHERING

Applications that write output files in each of the hosts involved in the execution may need to fetch all those files to transfer them back to the user once the execution is finished. The following example copies all the `mydata.*` files to the first host. It uses the `mpi_start_foreach_host` function of mpi-start that will call the first argument for each of the hosts passing the name of the host as parameter.

```
# the first paramter is the name of a host in the
my_copy () {
    CMD="scp . \$1:\$PWD/mydata.*"
    echo \$CMD
}

post_run_hook () {
    echo "post_run_hook called"
    if [ "x\$MPI_START_SHARED_FS" = "x0" ] ; then
        echo "gather output from remote hosts"
        mpi_start_foreach_host my_copy
    fi
    return 0
}
```

3.5 HOOKS VARIABLE SUMMARY

This section contains a summary of the variables that can modify the existing hook behaviour. They can be set using the `-d` command line switch.

Hook	Variable	Meaning
File Distribution	`MPI_SHARED_FS`	If undefined, mpi-start will try to detect a shared file system in the execution directory. If defined and equal to 1, mpi-start will assume that the execution directory is shared between all hosts and will not try to copy files. Any other value will make mpi-start assume that the execution directory is not shared.
File Distribution	`I2G_MPI_FILE_DIST`	Forces the use of a specific distribution method.
File Distribution	`MPI_SHARED_HOME`	If set to "`yes`", mpi-start will use the path defined in `MPI_SHARED_HOME_PATH` for copying the files and executing the application.
File Distribution	`MPI_SHARED_HOME_PATH`	Path to a shared directory.
File Distribution	`MPI_START_DISABLE_CLEANUP`	If set to "`yes`", mpi-start will not try to cleanup files after job execution.
Affinity	`MPI_USE_AFFINITY`	If set to 1, enable processor affinity hook.
OpenMP	`MPI_USE_OMP`	If set to 1, enable Open MP hook.
MPItrace	`I2G_USE_MPITRACE`	If set to 1, enable MPItrace hook.
Marmot	`I2G_USE_MARMOT`	If set to 1, enable Marmot hook.
Compiler	`MPI_COMPILER_HOOK`	If set to 0, disable compiler hook.

4 SYSTEM ADMINISTRATOR GUIDE

4.1 INSTALLATION

4.1.1 BINARY DISTRIBUTION

Binary packages for mpi-start are generated in the openSUSE build service and distributed by EMI. Check their repositories for the correct package for your distribution. Once you have the repositories configured you only need to install the package using your favorite package manager:
For RedHat based distributions:

```
yum install mpi-start
```

Debian based:

```
apt-get install mpi-start
```

If you are running a site with CREAM and WN, you may prefer to install the emi-mpi meta-package that includes the yaim plugin for configuraton:

```
yum install emi-mpi
```

The nodes where the user applications will be executed (Worker Nodes) also require a working MPI implementation, Open MPI and MPICH are recommended. The devel packages should also be installed in order to allow user to compile their applications. Refer to your OS repositories for the exact packages. In the case of SL5, Open MPI (including devel packages) can be installed with the following command line:

```
yum install openmpi openmpi-devel
```

devel packages may require also the installation of a C/C++/Fortran compiler. Some devel packages of the MPI packages do not include the compiler as (e.g. gcc, gcc-gfortran, gcc-g++) dependency! They should be installed also if you want to support the compilation of MPI applications.

4.1.2 UPGRADING FROM EMI-1

There are no major changes between EMI-1 and the EMI-2 releases, no backward incompatible changes have been introduced. However the metapackage has changed the name from `glite-mpi` to `emi-mpi`. If you configure the EMI-2 repo on top of a EMI-1 installation and do an upgrade, the glite-mpi package will be automatically upgraded to emi-mpi when a `yum upgrade` or `yum update` is performed. There is no need to reconfigure.

4.1.3 UPGRADING FROM EMI-2

The mpi-start packages are automatically upgraded when using the EMI-3 repositories, there is no need to reconfigure.

4.1.4 SOURCE DISTRIBUTION

Source can be retrieved from the git repository.

Installation is as easy as "./configure && make install". The default installation prefix is "/usr". If a non default installation wants to be done, use the PREFIX variable in make install

```
$ make install  PREFIX=/opt/mpi-start
```

In this case, is recommendable setting the installation the environment variable I2G_MPI_START to point to mpi-start script (although this is not mandatory anymore).

```
$ export I2G_MPI_START=/opt/mpi-start/bin/mpi-start
```

4.2 CONFIGURATION

mpi-start is designed to auto detect most of the site configurations without any administrator intervention. The default installation will automatically detect:

- the batch scheduler at the site: currently PBS/Torque, SGE, LSF, Condor and Slurm are supported.
- existence of shared file system in the job running directory

mpi-start uses a set of files to configure its behavior. There are several paths where the files can be located. All of them will be checked when looking for hooks, execution environments or scheduler plugins. These are the paths (and the order) used by default in mpi-start:

- Any path pointed by environment variable `MPI_START_ETC`.
- A `.mpi-start` directory at current user's home.
- The `etc/mpi-start` under mpi-start installation path. On default installations that would be `/`.

The first file that mpi-start checks is the `mpi-config.local` file. This should contain the appropriate location of your local MPI installations and any other modifications you want to introduce in the default behavior. mpi-start includes such file with the default configuration for your system in RHEL/SL 5, RHEL/SL 6 and Ubuntu.

Typical variables that the administrator can set in this file are:

Variable	Meaning
`MPI_DEFAULT_FLAVOUR`	name of the default flavour for jobs running at the site
`MPI_<flavour>_PATH`	Path of the bin and lib directories for the MPI flavour
`MPI_<flavour>_MODULE`	Name of the module that loads the MPI flavour environment. When defined, the `MPI_<flavour>_PATH` will not be used.
`MPI_<flavour>_VERSION`	preferred version of the MPI flavour
`MPI_<flavour>_MPIEXEC`	Path of the MPIEXEC binary for the specific flavour
`MPI_<flavour>_MPIEXEC_PARAMS`	Parameters for the MPIEXEC of the flavour
`MPI_<flavour>_MPIRUN`	Path of the MPIRUN binary for the specific flavour
`MPI_<flavour>_MPIRUN_PARAMS`	Parameters for the MPIRUN of the flavour
`MPI_<flavour>_MPI<compiler>`	Location of the compiler for the flavour. Compiler may be one of `CC`, `F90`, `F77` or `CXX`.
`I2G_<flavour>_PREFIX`	Path of the MPI installation for the MPI flavour

A known issue with the setting of the I2G_<flavour>_PREFIX variable makes them useless, please use the MPI_<flavour>_PATH variable instead!

If `MPI_<flavour>_MPIEXEC` or `MPI_<flavour>_MPIRUN` are not defined, mpi-start will try to use the mpiexec or mpirun that are found in the current PATH.

4.2.1 HOOKS

Hooks may change the behavior of mpi-start and provide additional features such as file distribution and configuration of compiler flags. Site admins can add their own hooks via the local hook mechanism.

mpi-start includes hooks for distributing the files needed for the execution of an application. By default it tries to find the most suitable method for copying the files, using shared filesystems whenever they are found. However, the filesystem detection may not work for all sites, or the shared filesystem may be in a different location to the execution path making it impossible for mpi-start to detect its availability. Check Section 3 for more information. Section 3.5 contains a summary of relevant variables that may defined.

4.3 MPI-START YAIM CONFIGURATION

Configuration is necessary on both the CE and WNs in order to support and advertise MPI correctly. This is performed by the yaim MPI module which should be run on both types of nodes.

4.3.1 WN CONFIGURATION

The yaim plugin in the WN prepares the environment for the correct execution of mpi-start. Each of the MPI flavours supported by the site must be specified setting the variable `MPI_<FLAVOUR>_ENABLE` to `"yes"`. For example, to enable Open MPI, add the following:

```
MPI_OPENMPI_ENABLE="yes"
```

Optionally, if you are using a non OS provided MPI implementation, you can define the location and version with `MPI_<FLAVOUR>_VERSION` and `MPI_<FLAVOUR>_PATH`. **Do not use these variables if you are using the OS provided MPI implementations**. For example for Open MPI version 1.3, installed at /opt/openmpi-1.3:

```
MPI_OPENMPI_VERSION="1.3"
MPI_OPENMPI_PATH="/opt/openmpi-1.3/"
```

MPI flavours that use a particular mpiexec for starting the jobs (e.g. OSC mpiexec for PBS/Torque system) may also provide in the `MPI_<FLAVOUR>_MPIEXEC` the path to the binary. **Do not use this variable if you are not using a different mpiexec from the one provided by the MPI implementation.**

Additionally, you may specify a default MPI flavour to use if non is selected for execution, with the `MPI_DEFAULT_FLAVOUR`. If no default flavour is specified, the first one defined in your site-info.def will be considered as default.

If you provide a shared filesystem for the execution of the applications, but it is not the path where the jobs are started, then set the variable `MPI_SHARED_HOME` to `"yes"` and the variable `MPI_SHARED_HOME_PATH` to the the location of the shared filesystem. **Do not use this variable if the application starts its execution in a shared directory (e.g. shared home), this situation should be automatically detected**.

If you use ssh host based authentication, set the variable `MPI_SSH_HOST_BASED_AUTH` to `"yes"`. **Note that this does NOT configure passwordless SSH between the nodes, just sets an environmnet variable**

Lastly, if your use a non default location for mpi-start, set its location with the `MPI_MPI_START` variable.

The complete list of configuration variables for the WN is shown in the next table:

Variable	Mandatory	Description
`MPI_<FLAVOUR>_ENABLE`	YES	set to `"yes"` if you want to enable the <flavour>
`MPI_<FLAVOUR>_VERSION`	NO	set to the supported version of the <flavour>, usually is automatically detected
`MPI_<FLAVOUR>_PATH`	NO	set to the path of supported version of the <flavour>, usually is automatically detected by the yaim WN plugin
`MPI_<FLAVOUR>_MPIEXEC`	NO	If you are using OSC mpiexec (only in PBS/Torque sites), set this to the location of the mpiexec program, e.g. `"/usr/bin/mpiexec"`
`MPI_DEFAULT_FLAVOUR`	NO	Set it to the default flavour for your site, if undefined, the first defined flavour will be used
`MPI_SHARED_HOME`	NO	set this to `"yes"` if you have a shared home area between WNs.
`MPI_SHARED_HOME_PATH`	NO	location of the shared area for execution of MPI applications
`MPI_SSH_HOST_BASED_AUTH`	NO	set it to `"yes"` if you have SSH based authentication between WNs
`MPI_MPI_START`	NO	Location of mpi-start if not installed in standard location (`/usr/bin/mpi-start`)

The profile for a worker node is MPI_WN. Use it along any other profiles you may need for your WN.

```
/opt/glite/yaim/bin/yaim -c -s site-info.def  -n MPI_WN -n <other_WN_profiles>
```

4.3.2 CE CONFIGURATION

As with the WN, individual flavours of MPI are enabled by setting the `MPI_<FLAVOUR>_ENABLE` associated variable to `"yes"`. The version of the MPI implementation must also be specified with the variable `MPI_<FLAVOUR>_VERSION`, e.g. for configuring Open MPI version 1.3:

```
MPI_OPENMPI_ENABLE="yes"
MPI_OPENMPI_VERSION="1.3"
```

Possible flavours are:

OPENMPI for Open MPI

MPICH for MPICH-1

MPICH2 for MPICH-2

LAM for LAM-MPI

The use of shared homes should be announced also by setting the MPI_SHARED_HOME to "yes".

If you are using PBS/Torque, you can set the variable MPI_SUBMIT_FILTER to "yes" in order to enable the submission of parallel jobs in your system. The submit filter assumes that your Worker Nodes are correctly configured to publish in their status the ncpus variable with the number of available slots. If that's not true in your case, you may edit the file /var/torque/submit_filter in line 71 to fit your pbsnodes output. An example for using the np value is commented out in the file.

The complete list of configuration variables for the CE is shown in the next table:

Variable	Mandatory	Description
MPI_<FLAVOUR>_ENABLE	YES	set to "yes" if you want to enable the <flavour>
MPI_<FLAVOUR>_VERSION	YES	set to the supported version of the <flavour>, usually is automatically detected
MPI_START_VERSION	NO	set to the available mpi-start version. If not set, the yaim plugin will try to figure out the version by checking if mpi-start is installed.
MPI_SHARED_HOME	NO	set this to "yes" if you have a shared home area between WNs.
MPI_SUBMIT_FILTER	NO	Set this to "yes" to configure the submit filter for torque batch system that enables the submission of parallel jobs. The configuration assumes that torque path is /var/torque or TORQUE_VAR_DIR variable if defined.

The profile for configuring the CE is MPI_CE.

```
/opt/glite/yaim/bin/yaim -c -s site-info.def  -n MPI_CE -n <other_ce_profiles>
```

Batch system and MPI: The batch system may need extra configuration for the submission of MPI jobs. In PBS, you may use the automatic creation of the submit filter with the MPI_SUBMIT_FILTER variable. **Note:** any changes to the submit filter will be overwritten if yaim is re-run. In the case of SGE you need to configure a parallel environment. Check the documentation of your batch system for any further details.

For glite-yaim-mpi versions <= 1.1.11, the submit filter assumes that the pbsnodes -a output has the ncpus= field in the status line correctly set. If not, please change the submit filter like shown in this diff:

```
--- submit_filter       2012-01-20 11:19:48.000000000 +0100
+++ submit_filter.new   2012-01-20 11:19:21.000000000 +0100
@@ -68,8 +68,8 @@
          if (m/^\s*state\s*=\s*(\w+)/) {
              $state = ($1 eq "offline") ? 0 : 1;
          # This may be changed to fit your nodes description
-         # } elsif (m/^\s*np\s*=\s*(\d+)/) {
-         } elsif (m/^\s*status\s*=\s*.*ncpus=(\d+),/) {
+         } elsif (m/^\s*np\s*=\s*(\d+)/) {
+         # } elsif (m/^\s*status\s*=\s*.*ncpus=(\d+),/) {
              my $ncpus = $1;
              if ($state) {
                  if (defined($machines{$ncpus})) {
```

The default behaviour of the submit filter has changed in version 1.1.11 to use the "np=xx" parameter of the pbsnodes command output. Check the patch shown previously for the changes applied.

MPI_CE and other yaim profiles: The `MPI_CE` profile should be the first in the yaim configuration, otherwise the Glue variables will not be properly defined. This restriction may be removed in future versions.

mpi-start version: The yaim plugin will publish in the tags the mpi-start version if mpi-start is installed at the CE. If not installed you should define the `MPI_START_VERSION` with the version available at the WNs.

4.3.3 EXAMPLE CONFIGURATION

Here is an example configuration (with both CEs and WN variables!):

```
#----------------------------------
# MPI-related configuration:
#----------------------------------
# Several MPI implementations (or "flavours") are available.
# If you do NOT want a flavour to be installed/configured, set its variable
# to "no". Else, set it to "yes" (default). If you want to use an
# already installed version of an implementation, set its "_PATH" and
# "_VERSION" variables to match your setup (examples below).
#
# NOTE 1: the CE_RUNTIMEENV will be automatically updated in the file
# functions/config_mpi, so that the CE advertises the MPI implementations
# you choose here - you do NOT have to change it manually in this file.
# It will become something like this:
#
#   CE_RUNTIMEENV="$CE_RUNTIMEENV
#              MPI_MPICH
#              MPI_MPICH2
#              MPI_OPENMPI
#              MPI_LAM"
#
# NOTE 2: it is currently NOT possible to configure multiple concurrent
# versions of the same implementations (e.g. MPICH-1.2.3 and MPICH-1.2.7)
# using YAIM. Customize "/opt/glite/yaim/functions/config_mpi" file
# to do so.

MPI_MPICH_ENABLE="yes"
MPI_MPICH_VERSION="1.2.7p1"

MPI_MPICH2_ENABLE="yes"
MPI_MPICH2_VERSION="1.0.4"

MPI_OPENMPI_ENABLE="yes"
MPI_OPENMPI_VERSION="1.1"

MPI_LAM_ENABLE="yes"
MPI_LAM_VERSION="7.1.2"

# set Open MPI as default flavour
MPI_DEFAULT_FLAVOUR=OPENMPI

#---
# Example for using an already installed version of MPI.
# Setting "_PATH" and "_VERSION" variables will prevent YAIM
# from using the default OS packages.
```

```
# Just fill in the path to its current installation (e.g. "/usr")
# and which version it is (e.g. "6.5.9").
# DO NOT USE UNLESS A NON DEFAULT LOCATION IS USED
#---
# MPI_MPICH_PATH="/opt/mpich-1.2.7p1/"
# MPI_MPICH2_PATH="/opt/mpich2-1.0.4/"

# If you do NOT provide a shared home, set $MPI_SHARED_HOME to "no" (default).
#
# MPI_SHARED_HOME="yes"

#
# If you do NOT have SSH Hostbased Authentication between your WNs, set the below
# variable to "no" (default). Else, set it to "yes".
#
# MPI_SSH_HOST_BASED_AUTH="yes"


# If you use Torque as batch system, you may want to let the yaim plugin
# configure a submit filter for you. Uncomment the following line to do so
# MPI_SUBMIT_FILTER="yes"

#
# If you provide an 'mpiexec' for MPICH or MPICH2, please state the full path to
# that file here (http://www.osc.edu/~pw/mpiexec/index.php). Else, leave empty.
#
#MPI_MPICH_MPIEXEC="/usr/bin/mpiexec"
```

5 EXAMPLES

5.1 SIMPLE JOB

Simple job using environment variables:

```
#!/bin/sh
# IMPORTANT : This example script execute a
#             non-mpi program with Open MPI
#
export I2G_MPI_APPLICATION=/bin/hostname
export I2G_MPI_TYPE=openmpi

$I2G_MPI_START
```

Same example using command line parameters:

```
mpi-start -t openmpi /bin/hostname
```

5.2 JOB WITH USER HOOKS

```
#!/bin/sh
#
# MPI_START_SHARED_FS can be used to figure out if the current working
# is located on a shared file system or not. (1=yes, 0=no);
#
# The "mpi_start_foreach_host" function takes as parameter the name of
# another function that will be called for each host in the machine as
# first parameter.
# - For each host the callback function will be called exactly once,
#   independent how often the host appears in the machinefile.
# - The callback function will also be called for the local host.

# create the pre-run hook
cat > pre_run_hook.sh << EOF

pre_run_hook () {
    echo "pre run hook called "
    # - download data
    # - compile program

    if [ "x\$MPI_START_SHARED_FS" = "x0" ] ; then
        echo "If we need a shared file system we can return -1 to abort"
        # return -1
    fi

    return 0
}
EOF

# create the post-run hook
cat > post_run_hook.sh << EOF
# the first paramter is the name of a host in the
```

```
my_copy () {
    CMD="scp . \$1:\$PWD/mydata.1"
    echo \$CMD
    #\$CMD
    # upload data
}

post_run_hook () {
    echo "post_run_hook called"
    if [ "x\$MPI_START_SHARED_FS" = "x0" ] ; then
        echo "gather output from remote hosts"
        mpi_start_foreach_host my_copy
    fi
    return 0
}
EOF

export I2G_MPI_APPLICATION=mpi_sleep
export I2G_MPI_APPLICATION_ARGS=0
export I2G_MPI_TYPE=openmpi
export I2G_MPI_PRE_RUN_HOOK=./pre_run_hook.sh
export I2G_MPI_POST_RUN_HOOK=./post_run_hook.sh

$I2G_MPI_START

# instead of the variable definition, the following command line could be used:
# mpi-start -t openmpi -pre ./pre_run_hook.sh -post ./post_run_hook.sh mpi_sleep 0
```

5.3 USING MPI-START WITH WMS

EMI provides the WMS software for submitting jobs to the different available resources. The WMS gets a job description in the JDL language and performs the selection and actual submission of the job into the resources on behalf of the user. The following sections describe how to submit a job using the WMS.

5.3.1 BASIC JOB SUBMISSION

Jobs are described with the JDL language. Most relevant attributes for parallel job submission are:

- CPUNumber: number of processes to allocate.
- Requirements: requirements of the job, will allow to force the selection of sites with mpi-start support.

The following example shows a job that will use 6 processes and it is executed with Open MPI. The `requirements` attribute makes the WMS to select sites that publish that they support mpi-start and Open MPI.

```
JobType       = "Normal";
CPUNumber     = 6;
Executable    = "starter.sh";
Arguments     = "OPENMPI hello_bin hello arguments";
```

```
InputSandbox  = {"starter.sh", "hello_bin"};
OutputSandbox = {"std.out", "std.err"};
StdOutput     = "std.out";
StdError      = "std.err";
Requirements  = member("MPI-START", other.GlueHostApplicationSoftwareRunTimeEnvironment)
                && member("OPENMPI", other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

The `Executable` attribute is a script that will invoke mpi-start with the correct options for the execution of the user's application. We propose a generic wrapper that can be used for any application and MPI flavour that gets in the `Arguments` attribute:

- Name of mpi-start execution environment (I2G_MPI_FLAVOUR variable), in the example: OPEN-MPI
- Name of user binary, in the example: hello_bin
- Arguments for the user binary, in the example: hello arguments

This is the content of the wrapper:

```
#!/bin/bash
# Pull in the arguments.
MPI_FLAVOR=$1

MPI_FLAVOR_LOWER=`echo $MPI_FLAVOR | tr '[:upper:]' '[:lower:]'`
export I2G_MPI_TYPE=$MPI_FLAVOR_LOWER

shift
export I2G_MPI_APPLICATION=$1

shift
export I2G_MPI_APPLICATION_ARGS=$*

# Touch the executable, and make sure it's executable.
touch $I2G_MPI_APPLICATION
chmod +x $I2G_MPI_APPLICATION

# Invoke mpi-start.
$I2G_MPI_START
```

User needs to include this wrapper in the `InputSandbox` of the JDL (`starter.sh`) and set it as the `Executable` of the job. Submission is performed as any other job:

```
$ glite-wms-job-submit -a hello-mpi.sh

Connecting to the service https://gridwms01.ifca.es:7443/glite_wms_wmproxy_server


==================== glite-wms-job-submit Success ====================

The job has been successfully submitted to the WMProxy
Your job identifier is:
```

```
https://gridwms01.ifca.es:9000/8jG3MUNRm-ol7BqhFP5Crg

==========================================================================
```

Once the job is finished, the output can be retrieved:

```
$ glite-wms-job-output https://gridwms01.ifca.es:9000/8jG3MUNRm-ol7BqhFP5Crg

Connecting to the service https://gridwms01.ifca.es:7443/glite_wms_wmproxy_server

==========================================================================

                        JOB GET OUTPUT OUTCOME

Output sandbox files for the job:
https://gridwms01.ifca.es:9000/8jG3MUNRm-ol7BqhFP5Crg
have been successfully retrieved and stored in the directory:
/gpfs/csic_projects/grid/tmp/jobOutput/enol_8jG3MUNRm-ol7BqhFP5Crg

==========================================================================


$ cat /gpfs/csic_projects/grid/tmp/jobOutput/enol_8jG3MUNRm-ol7BqhFP5Crg/std.*
Hello world from gcsic054wn. Process 3 of 6
Hello world from gcsic054wn. Process 1 of 6
Hello world from gcsic054wn. Process 2 of 6
Hello world from gcsic054wn. Process 0 of 6
Hello world from gcsic055wn. Process 4 of 6
Hello world from gcsic055wn. Process 5 of 6
```

5.3.2 MODIFYING MPI-START BEHAVIOR

mpi-start behavior can be customized by setting different environment variables. If using the generic wrapper, one easy way of customizing mpi-start execution is using the `Environment` attribute of the JDL. The following JDL adds debugging to the previous example by setting the `I2G_MPI_START_VERBOSE` and `I2G_MPI_START_DEBUG` variables to 1:

```
JobType       = "Normal";
CPUNumber     = 6;
Executable    = "starter.sh";
Arguments     = "OPENMPI hello_bin hello arguments";
InputSandbox  = {"starter.sh", "hello_bin"};
OutputSandbox = {"std.out", "std.err"};
StdOutput     = "std.out";
StdError      = "std.err";
Requirements  = member("MPI-START", other.GlueHostApplicationSoftwareRunTimeEnvironment)
             && member("OPENMPI", other.GlueHostApplicationSoftwareRunTimeEnvironment);
Environment   = {"I2G_MPI_START_VERBOSE=1", "I2G_MPI_START_DEBUG=1"};
```

Use of hooks is also possible using this mechanism. If the user has a file with the mpi-start hooks called `hooks.sh`, the following JDL would add it to the execution (notice that the file is also added in the `InputSandbox`):

```
JobType       = "Normal";
CPUNumber     = 6;
Executable    = "starter.sh";
Arguments     = "OPENMPI hello_bin hello arguments";
InputSandbox  = {"starter.sh", "hello_bin", "hooks.sh"};
OutputSandbox = {"std.out", "std.err"};
StdOutput     = "std.out";
StdError      = "std.err";
Requirements  = member("MPI-START", other.GlueHostApplicationSoftwareRunTimeEnvironment)
             && member("OPENMPI", other.GlueHostApplicationSoftwareRunTimeEnvironment);
Environment   = {"I2G_MPI_PRE_RUN_HOOK=hooks.sh", "I2G_MPI_POST_RUN_HOOK=hooks.sh"};
```

6 MPI-START INTERNALS

This section documents the internal variables of mpi-start. They might be used for development of hooks or configuration of mpi-start. As the documentation improves, these may be moved to other sections of the document.

6.1 GLOBAL CONFIGURATION VARIABLES

Variable	Default	Description
MPI_START_DUMMY_SCHEDULER	1	Enables or disables the dummy scheduler.
I2G_MPI_START_KEEP_FILES	0	Enables or disables the removal of temporary files at the end of execution.
I2G_MPI_START_FULL_TRACE	-	Enables or disables full trace of mpi-start.
MPI_START_DO_NOT_USE_WRAPPER	-	Enables or disables the use of a wrapper for the executable
MPI_START_SOCKETS	-	Number of sockets in the host (if not defined, mpi-start tries to detect them).
MPI_START_COREPERSOCKET	-	Number of cores per socket in the host (if not defined, mpi-start tries to detect them).
MPI_START_COREPERSOCKET	-	Number of cores per socket in the host (if not defined, mpi-start tries to detect them).
I2G_MPI_START_ENABLE_TESTING	-	If equal to "TEST", then do not call the main function. Used for sourcing the mpi-start file.

6.2 SCHEDULER PLUGIN VARIABLES

Variable	Description
MPI_START_SCHEDULER	Name of the scheduler.
MPI_START_HOSTFILE	File containing one line per slot available.
MPI_START_MACHINEFILE	File containing one line per host available.
MPI_START_HOST_SLOTS_FILE	File containing one line with a name of host, and the number of slots available in that host.
MPI_START_NP	Total number of processors to use.
MPI_START_NPHOST	Number of processes per host, may be undefined if slot allocation is used.

.

6.3 MPI EXECUTION VARIABLES

Variable	Description
MPIEXEC	Defined by each flavour, mpiexec executable
MPI_GLOBAL_PARAMS	Global parameters for mpiexec
MPI_LOCAL_PARAMS	Local parameters for mpiexec
MPI_START_SSH_AGENT	User (or system) specified ssh agent (used mostly by condor).
MPI_START_DISABLE_LRMS_INTEGRATION	If set to "yes", do not use any LRMS integration available in the MPI flavour.
MPI_MPICH2_DISABLE_HYDRA	If set to 1, disable the use of hydra launcher for mpich2.
OSC_MPIEXEC	Set to 1 if OSC mpiexec is found.
HYDRA_MPIEXEC	Set to 1 if hydra mpiexec is found.
OPENMPI_VERSION_MAJOR	Set to Open MPI major version.
OPENMPI_VERSION_MINOR	Set to Open MPI minor version.
OPENMPI_VERSION_RELEASE	Set to Open MPI release version.

A CONFIGURATION OF BATCH SYSTEM

The batch system must be ready to execute parallel jobs (i.e. more than one slot is requested for a single job). Each batch system has its own specific ways of configuring such support.

Here you can find the instructions to manually configure different batch systems to execute MPI jobs.

A.1 TORQUE/PBS

Torque/PBS can be configured with the yaim module as described in previous sections. In order to configure manually you will need to edit (create it if it does not exist) your torque configuration file (`/var/torque/torque.cfg` or `/var/spool/pbs/torque.cfg`) and add a line containing:

```
SUBMITFILTER /var/torque/submit_filter.pl
```

Then download the `submit_filter.pl` from http://devel.ifca.es/rep/submit_filter.pl and put it in the above location.

This filter modifies the script coming from the submission, rewriting the `-l nodes=XX` option with specific requests, based on the information given by `pbsnodes` command.

The submit filter is crucial. Failing to use the submit filter translates in the job being submitted to only one node, where all the MPI processes are allocated too, instead of distributing the job across several nodes.

Warning: updates tend to rewrite torque.cfg. Check that the submit filter line is still there after performing an update

A.1.1 MAUI

Edit your configuration file (usually under `/var/spool/maui/maui.cfg`) and check that it contains the following line:

```
ENABLEMULTIREQJOBS TRUE
```

The `ENABLEMULTINODEJOBS` parameter must not be set to FALSE (if not specified is TRUE by default). These parameters allow a job to span to more than one node and to specify multiple independent resource requests.

The maui version provided as third party in EMI/UMD (maui-3.2.6p21-snap.1234905291.5.el5) has a bug that prevents the use of more than one WN when submitting a parallel job. See https://ggus.eu/ws/ticket_info.php?ticket=57828 for details. It is recommended to use newer versions of maui that do not have this problem.

A.2 SGE

Support for parallel jobs under SGE is enabled using *Parallel Environments*. You will need to configure at least one parallel environment in order to execute the jobs. Check the Parallel Environment documentation for more information. The CREAM Blah scripts will automatically use the available parallel environment if a job that requires more than one CPU is submitted.

In the following example a PE configuration is shown:

```
[root@ce ~]# qconf -sp mpi
pe_name            mpi
slots              4
user_lists         NONE
xuser_lists        NONE
start_proc_args    /bin/true
stop_proc_args     /bin/true
allocation_rule    $fill_up
control_slaves     TRUE
job_is_first_task  FALSE
```

A.3 PASSWORDLESS SSH (HOSTBASED AUTHENTICATION)

Depending on the MPI implementation used and if the site does not have a shared file system, passwordless ssh between the nodes may be required between the WN. If that's the case, make sure that any pool account can login from one WN to any other using ssh without showing any password prompt.

B INSTALLATION OF MPI IMPLEMENTATION

In order to execute MPI jobs, the site must support one of the multiple MPI implementations available. Most extended are Open MPI and MPICH2. OS distributions provide ready to use packages that fit most use cases. SL provides the following packages:

- `openmpi` and `openmpi-devel` for Open MPI.
- `mpich2` and `mpich2-devel` for MPICH2.
- `lam` and `lam-devel` for LAM

Installation of devel packages for the MPI implementation is recommended, since this will allow users to compile their applications at the site. Moreover the Nagios probes will try to compile a binary, thus not having a working compiler will make them fail. Note that the compiler may not be specified as dependencies of the `-devel` packages. Make sure that `gcc` and related packages are available.

Note also that not all the implementations support tight integration with the batch system. **Tight integration is required for proper accounting numbers.**

The MPI packages must be installed at the nodes that will execute the jobs (WN).

B.1 OPEN MPI

Open MPI support tight integration with several batch system, including Torque/PBS and SGE, that may require recompilation of the packages in order to get it. Tight integration allows proper accounting of resources used (CPU time) and better control of the jobs by the system, thus avoiding zombie processes if something goes wrong with the application. The following sections describe the SGE and PBS/Torque cases:

B.1.1 SGE

The SGE tight scheduler integration allows Open MPI to start the processes in the worker nodes using the native batch system utilities, thus providing better process control and accounting. SL5 packages already include support for SGE with the `openmpi` and `openmpi-devel` rpms. After Open MPI is installed, you should see one component named gridengine in the ompi_info output:

```
$ ompi_info | grep gridengine
                 MCA ras: gridengine (MCA v2.0, API v2.0, Component v1.4)
```

Check the Open MPI FAQ at http://www.open-mpi.org/faq/?category=building#build-rte-sge for more information.

B.1.2 TORQUE/PBS

In the case of Torque/PBS in SL5 you will need to compile the packages for your site. The Open MPI FAQ (http://www.open-mpi.org/faq/?category=building#build-rte-tm) includes instructions for doing so. You can adapt the SL5 packages to support Torque/PBS following these steps:

- Download and install Open MPI source rpm: http://ftp2.scientificlinux.org/linux/scientific/5x/SRPMS/vendor/openmpi-1.4-4.el5.src.rpm

```
$ rpm -Uvh http://ftp2.scientificlinux.org/linux/scientific/5x/SRPMS/vendor/openmpi-1.4-4.el5.src.
Retrieving http://ftp2.scientificlinux.org/linux/scientific/5x/SRPMS/vendor/openmpi-1.4-4.el5.src.
warning: /var/tmp/rpm-xfer.DAMscP: Header V3 DSA signature: NOKEY, key ID 192a7d7d
   1:openmpi                warning: user mockbuild does not exist - using root
warning: group mockbuild does not exist - using root
########################################### [100%]
warning: user mockbuild does not exist - using root
warning: group mockbuild does not exist - using root
warning: user mockbuild does not exist - using root
warning: group mockbuild does not exist - using root
warning: user mockbuild does not exist - using root
warning: group mockbuild does not exist - using root
warning: user mockbuild does not exist - using root
warning: group mockbuild does not exist - using root
warning: user mockbuild does not exist - using root
warning: group mockbuild does not exist - using root
warning: user mockbuild does not exist - using root
warning: group mockbuild does not exist - using root
```

- Modify the spec file to include Torque/PBS support:

```
--- openmpi.spec        2010-03-31 23:18:20.000000000 +0200
+++ openmpi.spec        2011-03-07 18:37:11.000000000 +0100
@@ -114,6 +114,7 @@
 ./configure --prefix=%{_libdir}/%{mpidir} --with-libnuma=/usr \
         --with-openib=/usr --enable-mpirun-prefix-by-default \
         --mandir=%{_libdir}/%{mpidir}/man %{?with_valgrind} \
+        --with-tm \
         --enable-openib-ibcm --with-sge \
         CC=%{opt_cc} CXX=%{opt_cxx} \
         LDFLAGS='-Wl,-z,noexecstack' \
```

- Install Torque/PBS development libraries:

```
$ yum install libtorque-devel
```

- Build the RPMs

```
$ rpmbuild -ba /usr/src/redhat/SPECS/openmpi.spec
```

- Install the resulting RPMs:

```
$ yum localinstall -nogpgcheck /usr/src/redhat/RPMS/x86_64/openmpi-*
```

- Check that the support for Torque/PBS is enabled:

```
$ /usr/lib64/openmpi/1.4-gcc/bin/ompi_info | grep tm
              MCA memory: ptmalloc2 (MCA v2.0, API v2.0, Component v1.4)
                 MCA ras: tm (MCA v2.0, API v2.0, Component v1.4)
                 MCA plm: tm (MCA v2.0, API v2.0, Component v1.4)
```

B.1.3 OPEN MPI WITHOUT TIGHT INTEGRATION

Open MPI can use rsh/ssh for starting the jobs if no tight integration is available. Jobs will run if you have passwordless ssh enabled between the WN, but the accounting figures will be incorrect.

B.2 MPICH2

MPICH2 can use several launchers for starting the processes:

- MPD, which uses rsh/ssh for starting the processes, so it will not produce correct accounting numbers.
- Hydra, which also uses rsh/ssh and should support tight integration with some batch systems.
- For PBS/Torque, OSC Mpiexec http://www.osc.edu/~djohnson/mpiexec/index.php which includes tight integration.

mpi-start is able to select the most appropriate one if found (Hydra is prefered over MPD)

B.2.1 MPD

MPD is available in all versions of MPICH2 and uses rsh/ssh to start the processes. It was the default starter for versions < 1.3. It uses a .mpd.conf file at the home directory, so it is necessary to provide a way to access the home directory from the WN (usual case)

B.2.2 HYDRA

Hydra is the new starter of MPICH2 and the default since version 1.3. It is designed to natively work with multiple daemons such as ssh, rsh, pbs, slurm and sge. Notice that not all versions support all the daemons!. The version included with SL5 **does NOT support pbs or sge**, therefore passwordless rsh/ssh between the nodes is mandatory.

sge support is included since version 1.3b1. pbs is included in version 1.5a1. If you want to have tight integration (i.e. accounting) with MPICH2 and one of these systems you may need to download and compile the packages at your site.

B.2.3 OSC MPIEXEC

OSC Mpiexec provides tight integration for PBS/Torque system. In order to use itwith mpi-start you will need to define the variable `MPI_MPICH2_MPIEXEC` pointing to its location.

C DISTRIBUTION OF BINARIES

The MPI binaries that users want to run need to be accessible on every node involved in an MPI computation (it is a parallel job after all). There are three main approaches:

C.1 SHARED HOME/OTHER SHARED AREA

By far the best option is to provide user homes hosted on a shared filesystem. This could either be a network filesystem (e.g. NFS) or a cluster filesystem (e.g. GPFS or Lustre). Then the MPI binary you transfer in the Sandbox (or compile up) on the first MPI node will automatically be available on all nodes. This is the normal mode of operation for MPI, and what MPI users will probably expect.

mpi-start checks if the working directory of the job is in a shared filesystem (nfs, gfs, afs, smb, gpfs and lustre are detected) and considers that if the filesystem is shared the binaries will be available without any further action in all the nodes involved in the execution.

In some cases, there is an available shared filesystem but the job does not start its execution there. Site admins can force mpi-start to use one directory as shared for transferring the job files to all nodes as described in the hooks section.

C.1.1 PASSWORDLESS SSH BETWEEN WNS

If you configure host-based authentication between worker nodes, then mpi-start can automatically replicate your binary to nodes involved in the computation. All the files in the working directory will be replicated, however, other needed files (e.g. data) may not be replicated, so this would have to be done manually (and would be slow for large data sets). Also it could open up the potential for users to subvert the normal resource management mechanisms by directly executing commands on nodes not allocated to them.

C.1.2 USE OSC MPIEXEC TO DISTRIBUTE FILES

This option is for sites with neither a shared filesystem nor passwordless ssh between WNs. If you have an mpiexec that can spawn the remote jobs using the LRMS native interface, you can use it to distribute the files. See http://www.osc.edu/~djohnson/mpiexec/index.php#Cute_mpiexec_hacks for the basic idea.

D INFORMATION SYSTEM

Sites may install different implementations (or flavours) of MPI. It is important therefore that users can use the information system to locate sites with the software they require. You should publish some values to let the world know which flavour of MPI you are supporting, as well as the interconnect and some other things. Everything related with MPI should be published as GlueHostApplicationSoftwareRunTimeEnvironment in the corresponding sections.

D.1 MPI-START SUPPORT

May include the version.

```
GlueHostApplicationSoftwareRunTimeEnvironment: MPI-START
GlueHostApplicationSoftwareRunTimeEnvironment: MPI-START-1.3.0
```

D.2 MPI FLAVOUR(S)

<MPI flavour>

This is the most basic variable and one should be advertised for each MPI flavour that has been installed and tested. Currently supported flavours are MPICH, MPICH2, LAM and OPENMPI.

Example:

```
GlueHostApplicationSoftwareRunTimeEnvironment: MPICH
GlueHostApplicationSoftwareRunTimeEnvironment: MPICH2
GlueHostApplicationSoftwareRunTimeEnvironment: LAM
GlueHostApplicationSoftwareRunTimeEnvironment: OPENMPI
```

D.3 MPI VERSION(S)

<MPI flavour>-<MPI version>

This should be published to allow users with special requirements to locate specific versions of MPI software.

Examples:

```
GlueHostApplicationSoftwareRunTimeEnvironment: OPENMPI-1.0.2
GlueHostApplicationSoftwareRunTimeEnvironment: MPICH-1.2.7
GlueHostApplicationSoftwareRunTimeEnvironment: MPICH-G2-1.2.7
GlueHostApplicationSoftwareRunTimeEnvironment: OPENMPI-1.0.2-ICC
```

D.4 MPI COMPILER(S) – OPTIONAL

<MPI flavour>-<MPI version>-<Compiler>

If <Compiler> is not published, then gcc suite is assumed.

D.5 INTERCONNECTS – OPTIONAL

MPI-<interconnect>

Interconnects: Ethernet, Infiniband, SCI, Myrinet

Example:

```
GlueHostApplicationSoftwareRunTimeEnvironment: MPI-Infiniband
```

D.6 SHARED HOMES

If a site has a shared filesystem for home directories it should publish the variable `MPI_SHARED_HOME`.

```
GlueHostApplicationSoftwareRunTimeEnvironment: MPI_SHARED_HOME
```