Research Report

Preparing the Worldwide LHC Computing Grid for MPI applications

Research Report for RP2
University of Amsterdam
MSc in System and Network Engineering

Class of 2005-2006

Richard de Jong, Matthijs Koot
{rjong,mrkoot}@os3.nl

27th June 2006

Abstract

This is the abstract.

Contents

List of Figures

Chapter 1

Introduction

As part of our Master of Science study in the field of System and Network Engineering at the University of Amsterdam, we have done research at CERN, Geneva [1], on enabling MPI-based parallel computing on the LHC Computing Grid (LCG). The research was performed on behalf of NIKHEF [2], Amsterdam, and was supervised by David Groep. The work was done in close cooperation with Louis Poncet from the IT Grid Deployment group at CERN.

1.1 The Worldwide LHC Computing Grid project

The Worldwide LHC Computing Grid, or *WLCG* is the project to create a grid to support the physicists that will analyze the data coming from the Large Hadron Collider, a particle accelerator. The section below is taken from the LCG Project Overview[1] [3]:

> "The Large Hadron Collider (LHC), currently being built at CERN near Geneva, is the largest scientific instrument on the planet. When it begins operations in 2007, it will produce roughly 15 Petabytes (15 million Gigabytes) of data annually, which thousands of scientists around the world will access and analyse. The mission of the LHC Computing Project is to build and maintain a data storage and analysis infrastructure for the entire high energy physics community that will use the LHC.
>
> The data from the LHC experiments will be distributed around the globe, according to a four-tiered model. A primary backup will be recorded on tape at CERN, the "Tier-0" centre of LCG. After initial processing, this data will be distributed to a series of Tier-1 centres, large computer centres with sufficient storage capacity for a large fraction of the data, and with round-the-clock support for the Grid.
>
> The Tier-1 centres will make data available to Tier-2 centres, each consisting of one or several collaborating computing facilities, which can store sufficient data and provide adequate computing power for specific analysis tasks. Individual scientists will access these facilities through Tier-3 computing resources, which can consist of local clusters in a University Department or even individual PCs, and which may be allocated to LCG on a regular basis.
>
> Discovering new fundamental particles and analysing their properties with the LHC accelerator is possible only through statistical analysis of the massive amounts of data gathered by the LHC detectors ATLAS, CMS, ALICE and LHCb, and detailed comparison with compute-intensive theoretical simulations. The goals of the LCG project include:
>
> - Developing different software components to support the physics application software in a Grid environment.

[1] The W in WLCG is added later than the quoted text was written, to denote the difference between the grid middleware software stack which was used to be called LCG, and the worldwide grid using this software

- Developing and deploying computing services based on a distributed Grid model.
- Managing users and their rights in an international, heterogeneous and non-centralized Grid environment.
- Managing acquisition, installation, and capacity planning for the large number of commodity hardware components that form the physical platform for the LCG."

The workflow for processing the LHC data is depicted in figure 1.1, taken from [4].

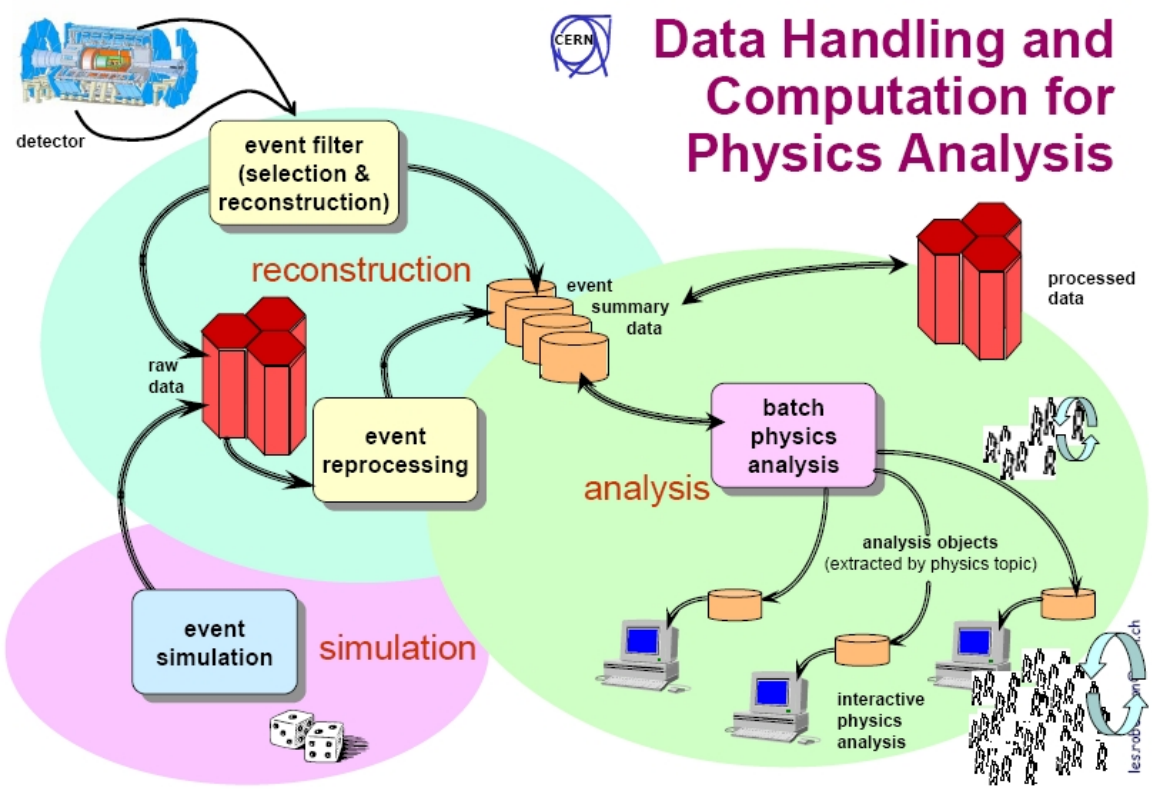

Figure 1.1: Data handling and computation for physics analysis

1.2 Grid

So, what exactly is a *grid*? According to Ian Foster [8], one of the 'fathers of the grid', a grid is a system that:

1. ... coordinates resources that are not subject to centralized control...
2. ... using standard, open, general-purpose protocols and interfaces...
3. ... to deliver nontrivial qualities of service.

Thus, it is more than simply a bunch of interconnected computational resources (which might rather be called a *cluster*); it includes social and political aspects as well. The most prevalent vision on grid computing is that of 'service-oriented computing', i.e. considering computational resources to be like water and electricity facilities [9, 10]. Exploitation of these means for scientific purposes is also denoted with the term *e-Science*, as coined by John Taylor [9]. Building on knowledge and code from Globus Toolkit, the European DataGrid project, or *EDG*, and the Enabling Grids for E-sciencE

project, or *EGEE*, the LCG project at CERN aims to deliver a production-quality world-wide grid for scientific purposes (and perhaps commercial usage at a later stage). It is believed that like every system needs an IP-address to be connected to the Internet, every system to be connected to 'the Grid' shall need to support the Open Grid Services Architecture, or *OGSA*. We will take OGSA into account where applicable.

During years of research on distributed computing, a general multilayer model has been developed for grid architectures, which is depicted in figure 1.2. This model is semantically comparable with the TCP/IP model used for Internet. The functions placed between the *User Applications* layer and the *Fabric* layer are provided by *Grid middleware*. In the case of the WLCG, that middleware is called *gLite*. gLite, which was previously unpractically and ambiguously named *EDG* and *LCG*, is a piece of software delivered by the JRA-1 group of EGEE. The gLite middleware is typically deployed through Yet Another Installation Method, or *YAIM*, an installation method developed in the course of Grid deployment. The OGSA specifies interfaces for all layers.

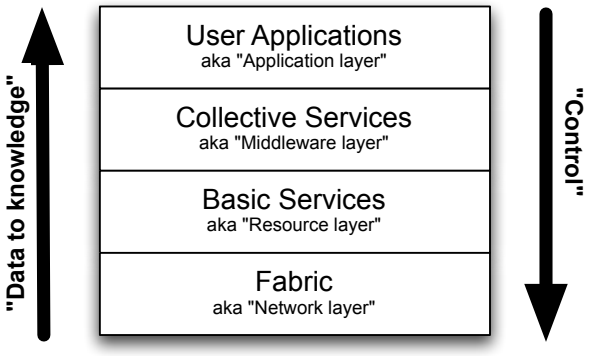

Figure 1.2: The layered grid model

1.2.1 Components of the Worldwide LHC Computing Grid

In LCG terminology, a Computing Element, or *CE*, is an abstraction of Worker Nodes, or *WNs*. CEs represent an interface to computing resources. One CE can be seen as a single cluster, owned by an organization that controls this cluster, defines who has access to it and, for instance, decides on scheduling decisions.

When an end-user submits a *job* described in the Job Description Language, or *JDL*, through a User Interface, or *UI*, the job is sent to a Resource Broker, or *RB*. The RB will notify the Logging & Bookkeeping component, or *LB*, and then query the Grid Index Information Service, or *GIIS*, to find CEs that are capable of executing the job. If such a CE is found, the RB delegates the job to it by accessing it through its *Gatekeeper*. The Gatekeeper then calls the CE Job Submission Agent, which distributes the job to the WNs.

1.3 Parallel computing

Parallel programming is the art of using multiple processors to solve a single problem. The traditional paradigm of computer architecture is the exact opposite: to solve multiple problems with a single processor (serial computing). With millions of scalar computers now connected through the Internet, the perspective on computing is slowly changing to that of 'the network is the computer'; instead of using a single processor to solve a problem, a global network of processors is now available to solve (the same or larger) problems. Historic methods of parallel computing included threading, Inter Process Communication (IPC) and Parallel Virtual Machines (PVM). None of them, however, are really suitable for a 'heavily distributed' environment like the globe-spanning WLCG [5]. In response

to this, a new protocol was designed to succeed PVM which has now, a decade later, become a *de facto* standard for massively parallel computing: the Message Passing Interface, or *MPI* [6]. MPI, which is a library specification, solves the problem of inter-process and job communication and allows data to be passed between processes in a distributed-memory environment. The prime quality attributes of MPI are source code portability, i.e. to support heterogeneous parallel architectures, and to allow efficient implementation, i.e. allow optimization for certain hardware platforms [7]. Although MPI may be used in shared-memory architectures (SMP, NUMA), its original design was focused on distributed memory architectures. It is the latter type of environment in which MPI is used at the WLCG; many scalar processors with their own memory, mostly grouped into scientific computing clusters, connected over the Internet.

1.4 Research goals

The research goals for our research were as follows:

Single-site MPI The primary goal of our research was to allow MPI-jobs to be submitted through the grid interface and be executed within a single CE. This consists of the following subgoals:

- Define the requirements for integrating MPI into the gLite middleware and the YAIM deployment scheme, so that, if enabled during gLite deployment, MPI jobs may be submitted through the WLCG grid interface (this includes scalability and dependency issues).
- Integrate (and demonstrate) MPI into the gLite middleware and the YAIM deployment scheme.

YAIM The secondary goal is to assess YAIM on elementary software quality attributes, as well as on adherence to best practices and standards that have been defined by the Global Grid Forum, or *GGF*[2].

Cross-site MPI The tertiary goal for our research was to pinpoint and document the issues that arise when trying to implement Cross-site MPI.

1.5 Acronyms

The following acronyms are used in this report:

API Application Programming Interface

BDII Berkeley Database Information Index

CE Computing Element

CERN European Laboratory for Particle Physics

EDG European DataGrid

EGEE Enabling Grids for E-sciencE

GGF Global Grid Forum

GIIS Grid Index Information Server

GRAM Globus Resource Allocation Manager

GRIS Grid Resource Information Service

GSI Grid Security Infrastructure

IS Information Service

JDL Job Description Language

LB Logging and Bookkeeping Service

LDAP Lightweight Directory Access Protocol

[2] GGF is often called 'the IETF of Grid computing'.

LHC Large Hadron Collider

LCG LHC Computing Grid

LRMS Local Resource Management System

LSF Load Sharing Facility

MPI Message Passing Interface

OS Operating System

PBS Portable Batch System

RB Resource Broker

RFIO Remote File Input/Output

SE Storage Element

SFT Site Functional Tests

SMP Symmetric Multi Processor

VDT Virtual Data Toolkit

VO Virtual Organization

WLCG Worldwide LHC Computing Grid

WMS Workload Management System

WN Worker Node

YAIM Yet Another Installation Mechanism

1.6 Copyrights and ownership

Chapter 2

Enabling MPI on the WLCG

2.1 Introduction to Parallel computing

Many computational problems are solvable through *serial computing*, i.e. using a single processor to solve a single problem. The problems which are dealt with in scientific and R&D communities often are not, though. Using a single scalar processor to solve large-scale problems, such as depicted in figure 2.2, is likely to exceed the acceptable amount of time to solve large problems (should be $< \Delta time$). In the Flynn-Johnson[1] taxonomy of computer systems [17], as depicted in figure 2.1, this kind of (non-parallel) system is classified as a Single Instruction, Single Data machine, or *SISD*. To handle a problem with multiple processors, one may divide it into smaller problems through *decomposition* and *parallel programming*. In *functional decomposition*[2], each processor is assigned a different role/responsibility, such as depicted in figure 2.3; this type of system is classified as Multiple Instruction, Single Data, or *MISD*. In *domain decomposition*[3], each processor is assigned a different part of data, such as depicted in figure 2.4; this type of system is classified as Multiple Instruction, Single Data, or *MISD*. In perfect domain decomposition, given a MISD with N processors, the total execution time may be decreased to $\Delta time/N$ (best case scenario). The last type of system in Flynn's original taxonomy, which dates back to 1972, is the Multiple Instruction, Multiple Data, or *MIMD* machine; an example is depicted in figure 2.5. In 1988, Johnson extended the taxonomy by differentiating the MIMD class on *Communication/Synchronization* and *Memory* attributes.

There are two main approaches to parallel programming [13]. In the *directives-based data-parallel language* approach, serial code is parallelized by adding directives that tell the machines how to distribute data and use multiple processors. Examples of such languages are OpenMP and High Performance Fortran. In the *message passing* approach, the approach relevant to our research, the programmers themselves are responsible for dividing data and work across the processors (striving for optimal load balancing), and manage communication between them (striving for minimal and non-blocking communication) by explicitly calling specific library functions.

Message passing already was an established paradigm for parallel computing back in the early nineties. Due to the proliferation of several incompatible libraries for parallel computing, the Message Passing Interface Forum, or *MPI Forum*, was established to provide for standardization. The group, consisting of some 60 people from 40 organizations, published version 1.0 of the Message Passing Interface standard in 1994 [14]. Version 1.1 was released in 1995, mainly to correct some errors and make clarifications in the first document. The most recent MPI is version 2.0, which was released in 1997 and is an extension (*not* a modification) to version 1.1. Today, MPI is the most widely-used standard for message passing on distributed-memory parallel computers.

[1] Other and more exhaustive approaches to classification of parallel systems exist, but the Flynn-Johnson appears most widely accepted and suffices for our purposes.

[2] Parallelism that arises through functional decomposition is also known as *task-parallelism*.

[3] Parallelism that arises through domain decomposition is also known as *data-parallelism*.

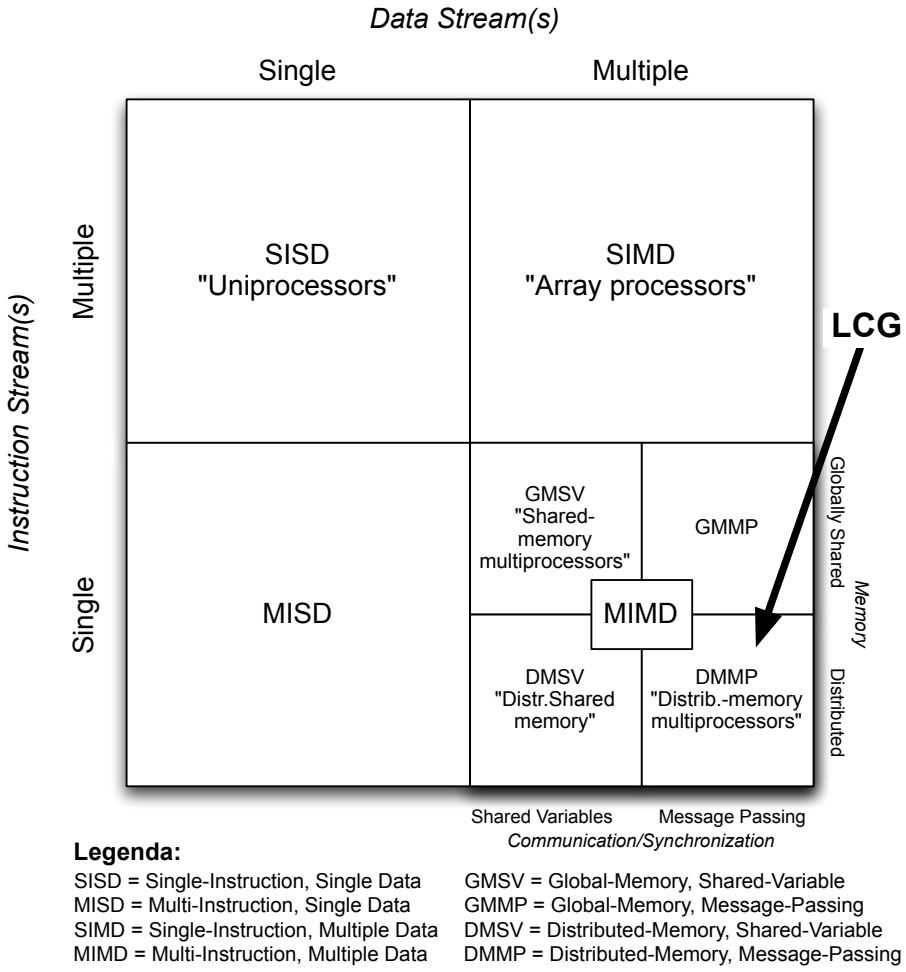

Figure 2.1: Flynn-Johnson taxonomy of computer systems.

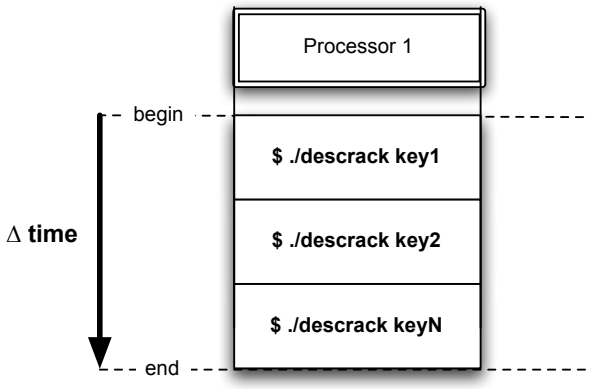

Figure 2.2: Serial computing: example of a SISD computation.

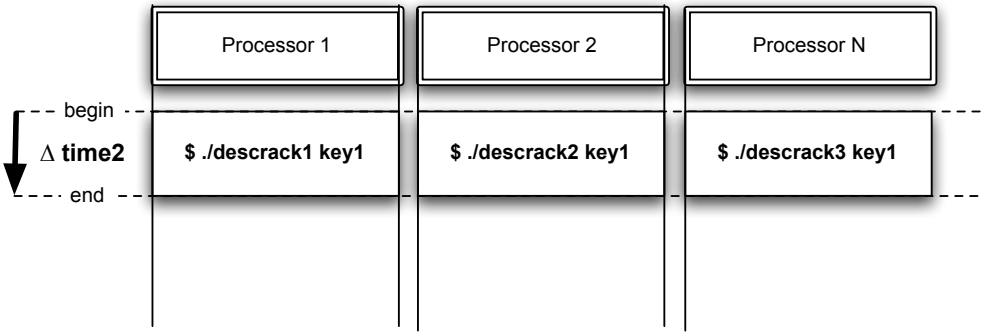

Figure 2.3: Parallel computing: example of a MISD computation.

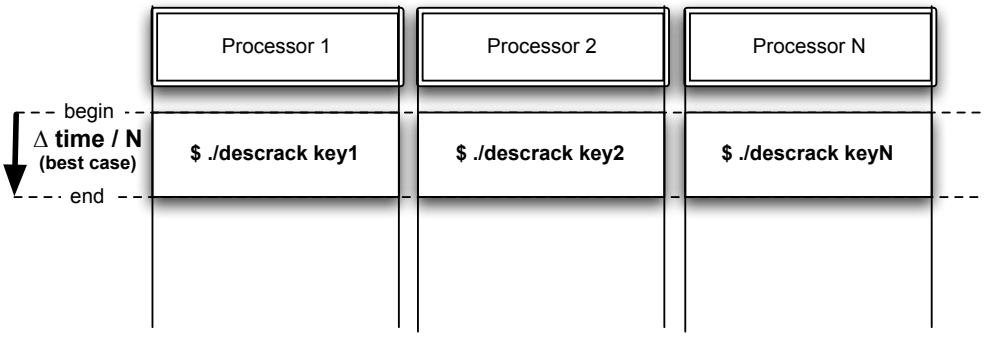

Figure 2.4: Parallel computing: example of a SIMD computation.

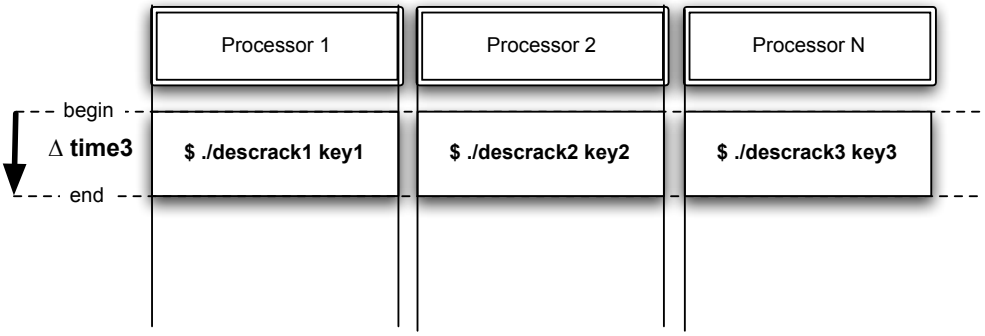

Figure 2.5: Parallel computing: example of a MIMD computation.

2.2 MPI on the WLCG, current situation

When discussing parallel computing on the WLCG, one is firstly discussing distributed-memory, message-passing MIMD-multiprocessors. However, a possible scenario would be to use distributed-memory for inter-CE computation and shared-memory for intra-CE or intra-WN computation (i.e. multiprocessor WNs or CEs with a high-speed inter-WN memory bus, such as Myrinet or Infiniband).

Currently, preliminary support for running MPI jobs on the WLCG is already available. It is possible to run an MPI job, provided the following conditions are met:

- SSH host-based authentication is set up between the Worker Nodes
- The attribute "MPICH" is advertised

When installing a Worker Node, the mpich package is installed. Mpich is one of the Open Source implementations of MPI.

The user can specify a Jobtype of "MPICH" in the job and request a number of nodes (WN's). This number of nodes is internally interpreted as a number of CPUs, it is currently impossible to specify both the required number of nodes and the number of processors.

The list of Resource Broker types that are capable of running MPI jobs is hard-coded in the gLite software.

The user needs to submit his job as a precompiled binary, linked to the same version of mpich as is installed on the CE his job is sent to. This leaves the user little flexibility in choice of an MPI implementation and in variations on it's version.

The user's application is wrapped inside a script that uses Secure Copy (SCP) to copy the executable to all the assigned hosts. Some sites prefer to use a shared filesystem and therefore do not need this. The wrapper script invokes the mpirun command, which does not integrate very well with the gLite software.

To conclude, the current solution for running MPI jobs is not complete, nor is it very flexible.

2.3 MPI on the WLCG, desired situation

Based on the experiences with MPI on the WLCG, in Q1 2006, a new working group was formed within the EGEE Technical Coordination Group (EGEE-TCG), specifically focussing on MPI, called EGEE-TCG-MPI [15]. This working group has elicited several requirements for MPI on the WLCG. They are cited in Appendix B. Together, these requirements form the desired situation.

In the next subsections, the relationship between those requirements and the LCG will be discussed. Facilitating MPI is neutral in some respects, but not in others; this mostly depends on whether only single-site MPI is facilitated, or users want to have their MPI jobs span multiple sites.

Most of the requirements are met when single-site MPI is deployed throughout the WLCG. To also meet the remaining requirements, cross-site MPI has to be facilitated. We propose to split up the effort in two phases: first enable single-site MPI, then enable cross-site MPI.

2.3.1 Single-site MPI (first phase)

Single-site MPI refers to the ability to submit an MPI job through the grid interface, and have it executed within a single CE (i.e. a single cluster).

2.3.1.1 Implications for the end-user

End-users will be able to execute MPI jobs by submitting them from a UI. MPI is a standardized interface specification, of which a whole breed of different implementations is available (in the dozens [12]). The main difference between these implementations is their prioritization of certain quality attributes; whereas one implementation focusses on sheer performance, others focus on portability and supportability, et cetera. Since MPI merely is an interface specification, code that behaves correctly when

linked against, say, OpenMPI, should behave correctly when linked against, say, MPICH2. Theoretically, anyway. In reality, the differences in implementation *may* lead to an 'unsafe' program that behaves differently on one system than it does on another [13]:

> *"Several outcomes are consistent with the MPI specification and the actual outcome depends on the precise timing of events. In order to complete successfully, an 'unsafe' program may require resources that are not guaranteed by the MPI implementation of the system on which it is running. Because of the varying MPI implementations, it is important to be aware of the issues you should consider when writing code that you want to be portable. Some of these issues are:*
>
> - ***Buffering Assumptions.*** *In standard mode, blocking sends and receives should not be assumed to be buffered. The reason for this is that buffer memory is finite and all computers will fail under sufficiently large communication loads. If you write a program using buffering assumptions, it will work under some conditions and fail under others. Hence, it is not portable.*
> - ***Barrier Synchronization Assumptions for Collective Calls.*** *In MPI, collective communications are always assumed to be blocking. However, your program should not depend on whether collective communication routines, like broadcast commands, act as barrier synchronizations. An MPI implementation of collective communications may or may not have the effect of barrier synchronization. One obvious exception to this is the MPI_BARRIER routine.*
> - ***Communication Ambiguities.*** *When writing a program, you should make sure that messages are matched by the intended receive call. Ambiguities in the communication specification can lead to incorrect or non-deterministic programs if race conditions arise. Use the message tags and communicators provided by MPI to avoid these types of problems."*

The end-users of the WLCG are advised to write MPI-code that is suitable for (tested with) an MPI-implementation that is (will be) explicitly provided by sites. If requirement `u3` is adequately fulfilled, this issue is unlikely to be an issue for most end-users; a range of different implementations and versions would then be available to them.

Requirement u1+u3+u5 implies that the user shall be responsible for calling `mpicc` on the WNs, as well as calling *the preferred implementation* of `mpicc`. This may be done by writing a Bash script in which `mpicc` is prefixed with an environment variable `$MPI_<implementation>_PATH`, in which `<implementation>` must refer to one of the CE-provided implementations of MPI (i.e. `$MPI_MPICH_PATH` for MPICH, as advertised by the CE). WNs shall be required to provide such variables if the CE advertises availability of MPI (one variable for each implementation), ref. section 2.3.1.3. The script is then assigned to the `Executable` parameter in the JDL-file, and the required source files are assigned to `InputSandbox`. Example usage:

```
#!/bin/sh
#
# MPIjob.sh
#

(...)

MY_MPI=$MPI_MPICH_PATH

$MY_MPI/bin/mpicc helloworld.c -o helloworld
$MY_MPI/bin/mpiexec -n 2 ./helloworld
(...)

/* job.jdl */
```

```
(...)
Executable = "MPItest.sh";
InputSandbox = "helloworld.c";
(...)
```

Requirement u2+s2 implies that if the user needs a shared directory, he will explicitly refer to that directory by referring to the `$MPI_PATH_TO_SHARED_HOME` environment variable. WNs shall be required to provide this variable if the CE advertises availability of a shared directory, ref. section 2.3.1.3. Example usage:

```
#!/bin/sh
#
# MPIjob.sh
#

(...)
echo `/bin/hostname` >> $MPI_PATH_TO_SHARED_HOME/unsafe-example.txt
(...)
```

Requirement s1 implies that the user shall pass the name of the preferred LRMS in the JDL file by means of specifying a restriction on the `GlueCEInfoLRMSType` attribute. Example usage:

```
/* job.jdl */
(...)
Requirements = other.GlueCEInfoLRMSType == "torque"
               || other.GlueCEInfoLRMSType == "lcgpbs"
               || other.GlueCEInfoLRMSType == "pbs"
               || other.GlueCEInfoLRMSType == "lsf";

(...)
```

2.3.1.2 Implications for RBs

Requirement u7 implies that an end-user is able to pass two MPI-related parameters to the RB, which denote the number of nodes required and the number CPUs to use on each node. As is, JDL specification v0.8 provides a single such parameter called *NodeNumber*, which may be assigned a positive numerical value. **Depending on how a JDL-job is submitted, this number indicates either the number of required 'nodes' (submission through Network Server**[4] **[19]) or the number of required CPUs (submission through WMProxy**[5] **[20])**. Due to interpretation issues, a *CPUNumber* parameter has been unofficially suggested, but its current status is unknown. Be it as it may, the RB will be required to explicitly process those parameter(s) while matching the job to available CEs. In gLite-3.0, this means that parts of these packages need to be modified: `org.glite.wms.helper`, `org.glite.wms.jdl`, `org.glite.wms.jdlj`, `org.glite.wms-ui.gui-java`.

2.3.1.3 Implications for CEs

The very fact that MPI needs to be facilitated has implications for the Local Resource Management Systems[6], or *LRMSs*, that are used by a CE. The implications are learned from [21]:

> *"Since in MPI, every process is a member of some communicator; when one allows MPI to create or destroy processes, all of the communicators that that process belongs to change. In order to keep collective operations on communicators meaningful (for example, what does*

[4] "The NodeNumber attribute is an integer greater than 1 specifying the number of nodes needed for an MPI job."
[5] "The NodeNumber attribute is an integer greater than 1 specifying the number of CPUs needed for an MPI job."
[6] *Local Resource Management Systems* are sometimes referred to as *Job Management Systems*, or *JMSs*, and several other synonyms have been seen in the wild; examples of such systems are OpenPBS, its successor Torque, and LSF.

a reduction mean when a process joins the reduction during the operation; for that matter, how is 'during' defined), all changes to communicators are collective operations."

Thus, to correctly manage an MPI job (e.g. provide for correct signal delivery, free resources, provide accounting), the LRMS and MPI implementation will need some coupling [22, 23, 24]. This integration may, for example, be done by having the LRMS offer an external interface to job scheduling functions and instruct the MPI library to use this API when spawning new processes. This approach has been demonstrated in the integration of LAM/MPI with OpenPBS [22] through the PSCHED API [31], which was an attempt to standardize such an external API. While this integration is still available in OpenMPI and Torque (the successors of respectively LAM-MPI and OpenPBS), PSCHED does not appear to be used by other LRMSs [30]. CEs that employ a LRMS that does not explicitly support parallelism cannot service MPI jobs and should therefore *never* advertise themselves as being MPI-capable.

Requirement u2 implies CEs need to explicitly advertise the (un)availability of a shared home directory. There is not yet an accepted way of advertising *availability of a shared home*, but this could be done by having the CE advertise "MPI_HOME_SHARED" (see instructions below). The exact path to the shared directory may not be known to the CE; therefore, as is, the LRMS wraps the job executable inside a preprocessing script that is run at the WNs and interprets such variables (i.e., `cd` to the right directory) and finalizes by initiating the actual job. *Unavailability of a shared home* is currently advertised as "MPI_HOME_NOTSHARED". This is done by modifying the `GlueHostApplicationSoftwareRunTimeEnvironment` value in the `/opt/lcg/var/gip/ldif/lcg-info-static-cluster.ldif` file (or rather, the `$CE_RUNTIMEENV` variable in your site-info.def and run `configure_node` again):

```
(...)
GlueHostApplicationSoftwareRunTimeEnvironment: MPI_HOME_NOTSHARED
(...)
```

The exact strings that may be advertised are *arbitrary*, and as is, coordination of naming conventions is up to the end-users; such responsibility belongs to the User Applications layer, of which the gLite middleware is no part.

Requirement u3+u5 implies CEs need to explicitly advertise the various MPI implementations they provide, i.e. by tagging themselves with {"OPENMPI-1.0.2", "MPICH-1.2.7", "MPICH-G2-1.2.7", ...}. This is done, again, by modifying the `GlueHostApplicationSoftwareRunTimeEnvironment` value in the `/opt/lcg/var/gip/ldif/lcg-info-static-cluster.ldif` file:

```
(...)
GlueHostApplicationSoftwareRunTimeEnvironment: OPENMPI-1.0.2
GlueHostApplicationSoftwareRunTimeEnvironment: MPICH-1.2.7
GlueHostApplicationSoftwareRunTimeEnvironment: MPICH-G2-1.2.7
(...)
```

Requirement u7 implies that an end-user is able to pass two parameters to the LRMS, instructing it to assign a certain number of nodes and the number of CPUs per node for a job. As is, JDL specification v0.8 provides a parameter called *NodeNumber*, which may be assigned a positive numerical value designating that required number of nodes [45]. Due to interpretation issues, a *CPUNumber* parameter has been unofficially suggested, but its current status is unknown. The current JDL specification does not provide for a way to fulfill requirement u7 as cited. Be it as it may, presuming that the JDL will provide for such parameters, the LRMS will be required to process those parameter(s). In gLite-3.0, this means that parts of these packages need to be modified: `org.glite.ce.blahp`, `org.glite.ce.cream`, `org.glite.cream-api-java`, `org.glite.ce.cream-cli`.

2.3.1.4 Implications for WNs

Requirement u1+u3+u5 implies that WNs shall provide working `mpicc` compiler and run-time environments for multiple implementations of both MPIv1 and MPIv2. The paths to these environments

shall be available for user scripts by having the WN provide a `$MPI_<implementation>_PATH` environment variable for each MPI-implementation, in which `<implementation>` is the name of the preferred implementation. For example, when OpenMPI-1.0.2 and MPICH-1.2.6 are supported, the WN shall execute:

```
#!/bin/sh
# some WN/bootscript

(...)
export MPI_MPICH_PATH=/opt/mpich-1.2.6
export MPI_OPENMPI_PATH=/opt/openmpi-1.0.2
(...)
```

Requirement u2 implies that WNs shall provide an environment variable called `$MPI_PATH_TO_SHARED_HOME` which contains the path of the shared directory. This is not required if the CE does not advertise the availability of a shared directory.

2.3.2 Cross-site MPI (second phase)

Cross-site MPI refers to the ability of submitting an MPI job through the grid interface, and have it executed *accross multiple CEs* (i.e. spanning a single MPI job over multiple clusters). Requirement u4+s3+s4+s5+s9 states that MPI jobs may indeed span multiple CEs. The implications of such requirements on the WLCG are discussed in the next sections.

2.3.2.1 More implications

Cross-site MPI adds several implications to those discussed for single-site MPI, and is a non-trivial requirement. Figure 2.6 depicts the key problem areas. The problems have been numbered by priority ('common sense'-style).

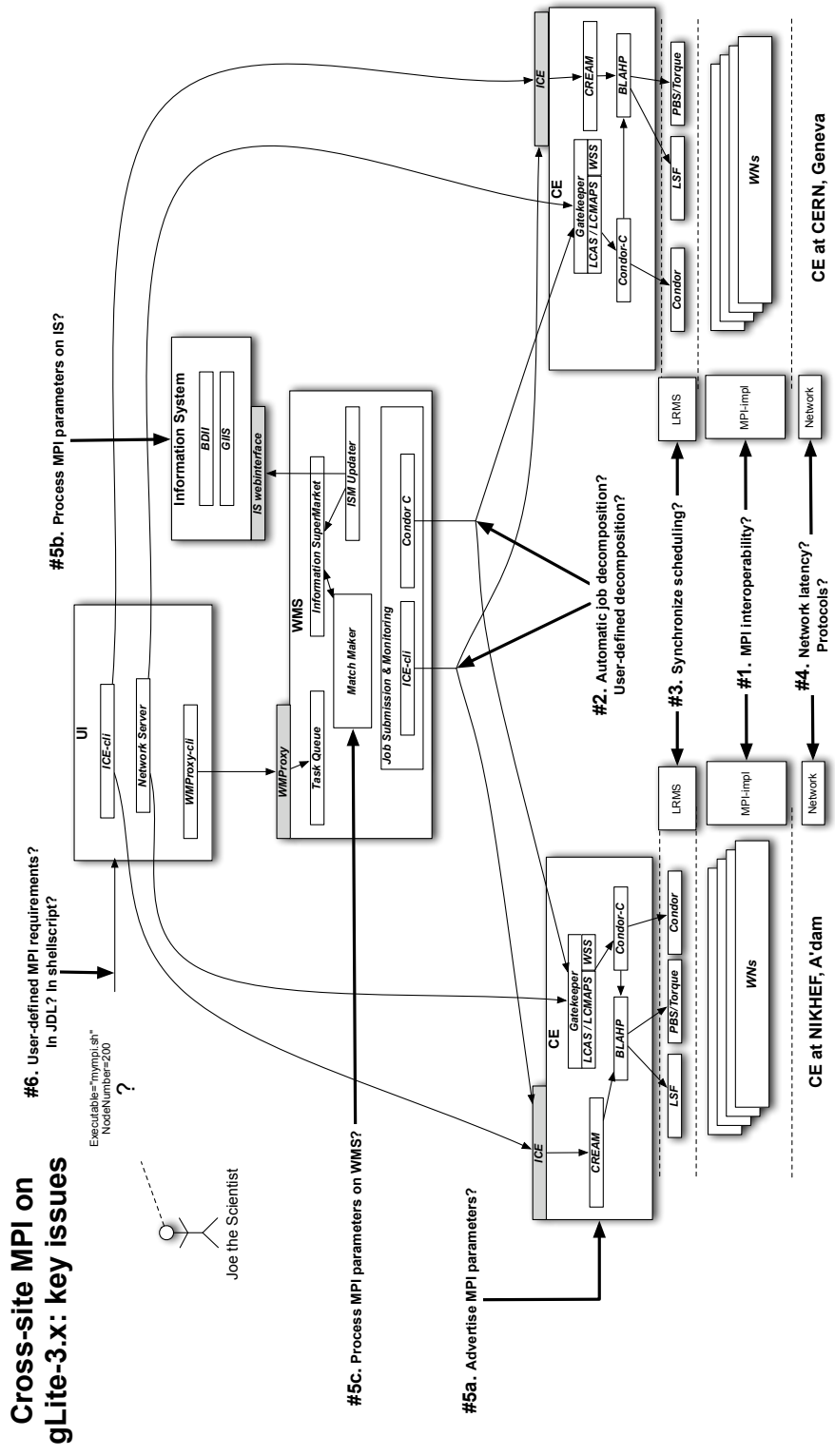

Figure 2.6: Key (problem) areas of cross-site MPI.

Problem #1 MPI interoperability?

Description By default, MPI implementations of different vendors are *not interoperable* (this also goes for the open-source ones). It is not possible to span a job over CEs that do not provide a common MPI implementation, as depicted in more detail in figure 2.7 [33]. An open, low-level protocol called *IMPI* has been suggested to solve the interoperability problem, which has (only) been implemented in LAM/MPI [32], WMPI II, HP MPI [34] and GridMPI [35]. More recently, HeteroMPI was suggested [36].

Solution 1 Require different sites to run the same implementation(s) of MPI.

Solution 2 Require different sites to run an IMPI-supporting implementation of MPI.

Solution 3 Initiate a working group to build yet another implementation of MPI that fulfills the grid-specific needs.

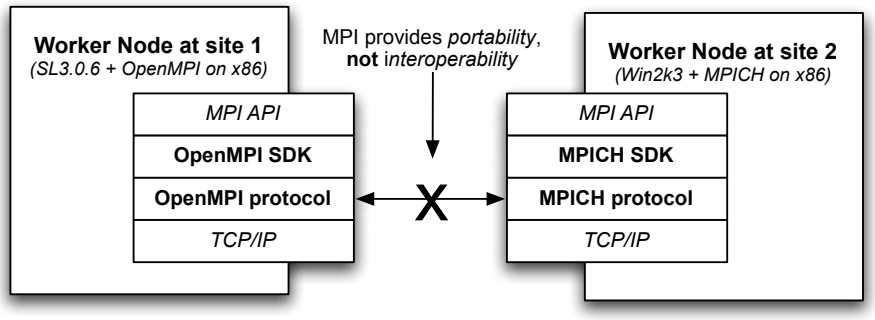

Figure 2.7: Run-time incompatibility between MPI implementations.

Problem #2 Automatic job decomposition? User-defined decomposition?

Description To span a single job over multiple CEs, the problem needs to be decomposed in a way that allows efficient fabric usage and is forgiving to network latency; i.e. independently schedulable jobs. Although it may currently be done for some classes of problems, manually, it is unclear how to do so in general — let alone how to automate it.

Solution Future work; this is area of current academic research. It seems that any solution to this problem will involve both end-users (i.e. MPI programmers) and grid middleware engineers. One relevant source is the MSc-thesis of Sean Philip Peisert in 2000, in which he proposed a programming model for automated decomposition on heterogeneous clusters [37].

Problem #3 Synchronize scheduling?

Description Jobs which cannot be decomposed in a way that completely avoids the need for inter-CE communication, require the LRMSs (or their abstractions) of the participating CEs to provide some form of synchronization and/or reservation mechanisms.

Solution Future work. It will include modifications to CE and/or LRMS components. A solution may come from analyzing the DUROC scheduling component of MPICH-G2 [29] and the KOALA grid scheduling system developed at TU Delft [38]. In KOALA, a concept called *Incremental Claiming Policy* is introduced to address the reservation problem, which "optimistically postpones the claiming of the processors for the job to a time close to the estimated job start time"; this has been demonstrated in the DAS-2 system.

Problem #4 Network latency? Protocols?

Description Latency is an important issue for MPI jobs. In [39], it is shown most MPI benchmarks only scale fine up to 20ms RTT; in real life, however, the latency might be up to hundreds of ms, depending on topological and geographical distance between sites participating in the job. Walking up the stack of the OSI networking model, another problem appears at the transport layer: it has been suggested that TCP is rather unsuitable as a carrier of MPI communication (TCP is designed for contiguous data, whereas MPI is typically non-contiguous).

Solution 1 It has been suggested that some three small modifications to TCP may improve its performance for MPI purposes. This includes the use of a *pacer* at start-up of an MPI job to prevent TCP from going into 'Slow Start' mode [40, 41], and switching TCP parameters at different stages of the MPI execution flow [41]. Applying these modifications may improve performance by 10% to 30% [41, 42];

Solution 2 It has been suggested that SCTP is more suitable for MPI than TCP [43]. Depending on message length and acceptable packet loss, using SCTP instead of TCP may increase MPI communication performance between 250% and 1100%;

Solution 3 Design latency-tolerance into parallel jobs (this is area of current academic research).

Problem #5a Advertise MPI parameters?

Problem #5b Process MPI parameters on IS?

Problem #5c Process MPI parameters on WMS?

Description The availability of MPI implementations and several MPI-specific parameters need to be processed throughout several grid components, including the WMS, the IS and the CE. The CE shall need to advertise the MPI implementations it supports and the (un)availability of a shared home directory to the IS. The WMS shall need to accept such parameters into its ISM, and the MatchMaker component shall need to be able to perform matchmaking on those MPI-specific requirements.

Solution See section 2.3.1; this is a topic for the JRA-1 group.

Problem #6 User-defined MPI requirements? In JDL? In shellscript?

Description The end-user shall need to specify the requirements to the MPI environment in the job description or the job executable.

Solution See section 2.3.1 and appendix D.

2.3.2.2 Prior art

Cross-site MPI has already been demonstrated in 2001 using the now-proprietary Legion MPI [26, 27], and in 2005 on TeraGrid using MPICH-G2 [28]. Similar environments are probably out there. MPICH-G2, a 'grid-enabling' extension to MPICH [29], in essence allows MPICH to walk the grid stack to the level needed to coordinate an MPI job over multiple sites, as depicted in figure 2.8. Its implementation is tightly coupled to the Globus Toolkit.

We strongly advise the WLCG team to consider MPICH-G2 as a pilot for facilitating cross-site MPI (assuming cross-site MPI indeed needs to be facilitated). MPICH-G2 has already been demonstrated to work with the Globus Toolkit 2.x grid service architecture, and full integration with the future webservices architecture is on its current roadmap [25]:

"COMING SOON: We have a version of MPICH-G2 that is fully integrated with the web services distributed in version v4.x of the Globus Toolkit. We are currently working on that version to complete its integration with the new MPICH-2 library and expect to have our first release in the first quarter of 2006"

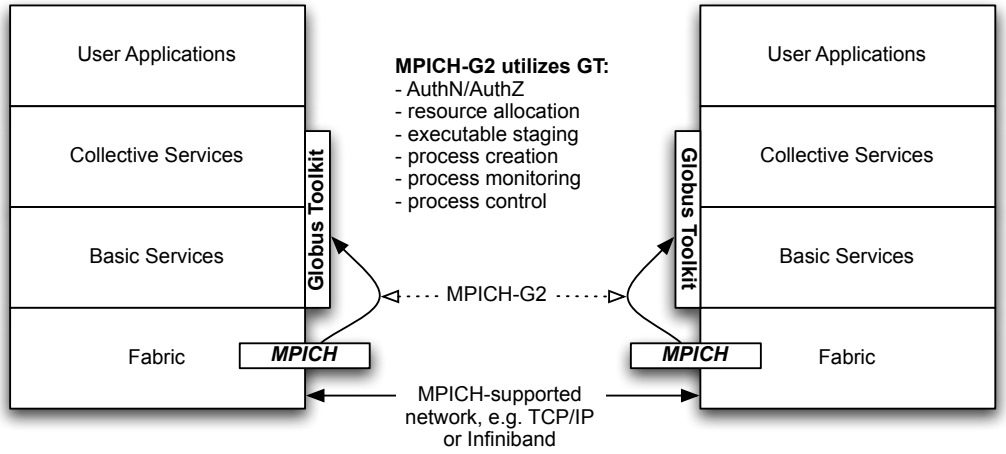

Figure 2.8: Cross-site MPI with MPICH and MPICH-G2

2.3.3 Remarks

Some of the implications discussed in the previous sections are reflected in YAIM code. Please refer to appendix C for a listing of the exact changes. The following section contains a listing of comments on the requirements which have not been (fully) addressed by us.

2.3.3.1 Unmet requirements

Requirement u2

Remark Scalability, reliability and availability of the shared directory are a responsibility of the site, and will not be addressed in our work.

Requirement u5

Remark Commercial/proprietary compilers are beyond the scope of our research due to anticipated mind-numbing software licensing issues.

Requirement s1+s7

Remark Due to the required integration (albeit loose or tight) between LRMS and MPI libraries and a lack of standardization in that respect, it is currently not feasible to fulfill this requirement. Instead, it is chosen to provide support for one major LRMS, namely Torque.

Requirement s6

Remark This requirement is being addressed through Savannah (bug-reporting) tickets and is the responsibility of the EGEE JRA-1 group.

Requirement s8

Remark This requirement is considered a 'recommendation' and has been used as such.

2.3.3.2 Efficiency of fabric usage

It is hard to estimate the performance of a message passing job and thus to know *ex ante* if the grid fabric will be unnecessarily stressed. This is actually true for *all* parallel programs, since the datasets they process might contain inefficiencies as well. It is the responsibility of the end-users to provide efficient code (i.e., do not use parallelism when you do not need to); global deployment of MPI will not by itself affect the efficiency of fabric usage on the WLCG.

2.4 Implementations of MPI and their fitness for the WLCG

A large number of implementations of MPI exist. They vary in a number of degrees. They may implement MPI-1, MPI-1 and MPI-2, or MPI-1 and a subset of MPI-2. It appears to be very difficult to implement MPI-2 completely, since only one of the implementations currently available offers full MPI-2 support.

There are different mechanisms for distributing the executable and starting the executable on the assigned machines. Most of the implementations rely on some form of remote shell access, for instance through the RSH or SSH commands. Most implementations use a daemon framework which can be set up before starting the job and can often remain active so that the next job can be run quickly.

Another issue is collaboration with the LRMS. Some have no integration with LRMS's, some provide only the TM interface that comes with Torque/OpenPBS, some support SLURM and/or SGE. Few implementations support all possible LRMS's. This is probably due to the fact that there is no standardized interface for applications to interact with LRMS's. There has been an effort to provide such a standardized interface, the PSCHED Task Management API (TM) [31], but it has not really caught on: only PBS with its derivatives support it with the TM interface.

If there is no collaboration between the MPI implementation and the LRMS, it is often still possible to submit and run jobs. Most LRMS's provide a list of hosts that are assigned to the job through an environment variable. This list can be used by the user's wrapper script, but there is no guarantee that he will not pick other hosts to use as well. This requires a lot of trust in the user who has to provide a wrapper script to read out this environment variable and start his application.

Collaboration with the LRMS is important, because it enables accounting of resource use and correct termination of processes: without collaboration, one host is assigned by the LRMS to start the application on the assigned hosts, and only this host is reported in the LRMS statistics. When there is collaboration, the LRMS takes care of starting the application on each of the hosts and therefore all hosts will be reported in the statistics.

We continue to describe a number of available Open Source implementations of MPI, that may or may not fulfill all the requirements, and describe how well they work in the WLCG.

2.4.1 LAM

LAM, which stands for Local Area Multicomputer, comes standard with Red Hat Enterprise Linux, and is therefore included in Scientific Linux, the Linux distribution created by CERN and Fermilab and other labs and universities to provide one standard base Operating System for various researchers [46]. It implements MPI-1.2 and much of MPI-2.

The project is in maintenance mode, because LAM is one of the contributing implementations to OpenMPI which is discussed in section 2.4.4. OpenMPI will obsolete LAM.

2.4.1.1 LAM-6.5.9

Version 6.5.9 of LAM is shipped with Red Hat ES 3. This version does not support any LRMS and is therefore not the best candidate for use within the WLCG. It offers the `lamboot` command to start up the LAM environment. It requires a list of hosts to which it logs in using SSH by default. It uses this to start a daemon under the current user account on all the hosts that listens on 2 ports, 32870 and 32871 for requests.

If this has succeeded, the `mpirun` command can be issued to start the MPI application on some or all of the hosts that have been set up. The `mpirun` command has the option of uploading (*staging*) the executable to all the required hosts before starting it. It does not have a solution for uploading the data to process.

When the application has finished running, the `lamhalt` command can be issued to stop all the daemons that have been started. For this SSH logins to all hosts are performed.

2.4.1.2 LAM-6.5.9 on the WLCG

Although not the best solution, it is possible to use LAM on the WLCG. LAM is installed easily by running something like:

```
apt-get install lam
```

on a Scientific Linux 3 machine. Running jobs on LAM requires the user to create a wrapper script to boot the LAM environment using the hosts provided by the LRMS. An example script can be found in Appendix E.1. It takes the name of the program to run on the commandline. It boots the LAM environment if necessary, compiles the .c file, runs the executable, and breaks down the LAM environment if required.

Setting the correct paths is not much of an issue in this case, because LAM is installed in the default system locations, such as `/usr/bin`, that are already in the path.

The script assumes SSH authentication has been set up between the Worker Nodes.

2.4.1.3 LAM-7.0.6

Version 7.0.6 is provided with Redhat ES 4, so it is also included in Scientific Linux 4. As of release 7, LAM supports TM (the PSCHED API), which makes it an interesting solution because now it integrates neatly with Torque/PBS, the default LRMS in the gLite Grid middleware.

2.4.1.4 LAM-7.0.6 on the WLCG

If a worker nodes runs Red Hat ES 4 or one of it's derivatives, it can simply be installed with the usual means, for instance by running

```
apt-get install lam
```

The integration with Torque/PBS should make it straightforward to start a LAM/MPI job, just run `lamboot` without any arguments. It will detect it is running from within Torque/PBS and will use `tm_spawn` function calls to set up the required daemons. Then an invocation of `mpirun` will use the hosts provided by Torque/PBS.

Setting the correct paths is not much of an issue in this case, because LAM is installed in the default system locations, such as `/usr/bin` that are already in the path.

2.4.1.5 LAM-7.1.2

The latest version of LAM is 7.1.2. On it's website [53] the RPM and SRPM (Source RPM) can be downloaded. The SRPM builds nicely on SLC3 (Scientific Linux 3, CERN edition) and can be installed with no trouble. A reminder here is that Grid middleware is not supposed to just overrule Operating System provided packages unless necessary. Therefore it would be a good idea to modify the spec file[7] in the SRPM, to build and install into a different directory, for instance in `/opt/lam-7.1.2`. Unfortunately, we had some trouble building this RPM due to some peculiarities in the spec file.

2.4.1.6 LAM-7.1.2 on the WLCG

As mentioned before, take care to not just upgrade components of the Operating System. If one manages to install LAM-7.1.2, the use of it is just as easy as described in section 2.4.1.4. Care must be taken however to determine the correct path.

[7] A spec file describes how to create an RPM from the provided tarball with the source code

2.4.2 MPICH

MPICH is a freely available, portable implementation of MPI-1, created by Argonne National Lab. It uses SSH to connect to remote hosts and launch the requested application. It offers multiple ways to communicate between processes. For a distributed system with TCP/IP communication between the nodes, `ch_p4` should be chosen.

MPICH has no notion of LRMS's, so it cannot cooperate with any of them. To work around this problem, Mpiexec was developed and should be used in Torque/PBS environments, see section 2.4.6.

2.4.2.1 MPICH-1.2.6 on the WLCG

MPICH-1.2.6 has been in the LCG middleware stack for quite some time, and is still included in the gLite 3.0 release. There is rudimentary support for MPI jobs available based on this software. It has been preconfigured to use the `ch_p4` device for communication.

One problem with this version of MPICH as provided in gLite, is that it installs in the normal system directories, just as LAM does. Therefore MPICH and LAM cannot co-exist without modifying one or the other. Since LAM is a default system component, it makes sense to modify the MPICH RPM so that it installs in another location. This has already been done for MPICH-1.2.7, see section 2.4.2.2. Therefore we propose to upgrade MPICH from 1.2.6 to 1.2.7, which will solve this problem.

Another problem with the current MPI support in gLite is that the invocation of `/usr/bin/mpirun` is hard coded in the jobwrapper template. Use of mpirun is not the preferred way of starting MPI applications. This should be fixed by removing the invocation of mpirun and instead let the user provide a script that executes mpiexec.

2.4.2.2 MPICH-1.2.7 on the WLCG

The latest version of MPICH is 1.2.7p1. An RPM has already been created by Charles Loomis of the Laboratoire de l Accélérateur Linéaire (LAL) [47] of the Institut national de physique nucléaire et de physique des particules (IN2P3) [48] in France and can be found in this directory [49]. This RPM installs the software in `/opt/mpich-1.2.7`, so it does not clash with a LAM installation. Therefore the path needs to be modified if a user wants to start an MPICH MPI job, by prepending the `$PATH` variable accordingly.

2.4.3 MPICH2

From the MPICH site of Argonne National Lab [50]:

> "MPICH2 is an all-new implementation of MPI, designed to support research into high-performance implementations of MPI-1 and MPI-2 functionality. In addition to the features in MPICH, MPICH2 includes support for one-side communication, dynamic processes, intercommunicator collective operations, and expanded MPI-IO functionality. Clusters consisting of both single-processor and SMP nodes are supported. With the exception of users requiring the communication of heterogeneous data, we strongly encourage everyone to consider switching to MPICH2. Researchers interested in using using MPICH as a base for their research into MPI implementations should definitely use MPICH2."

MPICH2 uses a different mechanism to start processes from that of MPICH, which is called MPD. This mechanism looks a lot like that of LAM. First the `mpdboot` command is used to start an `mpd` daemon on the hosts specified. This can be checked using the `mpdtrace` or `mpdringtest` commands. When the MPD-*ring* is set up, the `mpiexec` command can be given to run the command on all or some of the hosts. When the task is finished, the `mpdallexit` command is used to shut down all the daemons.

2.4.3.1 MPICH2 on the WLCG

The current version of MPICH2 is 1.0.3. It is distributed in .tar.gz format, but an RPM has been provided by Charles Loomis and can be found at [49]. This RPM installs itself in the `/opt/mpich2-1.0.3` directory, so it does not conflict with the other MPI implementations.

To be able to run MPICH2 jobs on the WLCG, wrapper scripts are provided in Appendix E.2 and E.3.

2.4.4 OpenMPI

From the OpenMPI website [51]:

> "Open MPI is a project combining technologies and resources from several other projects (FT-MPI, LA-MPI, LAM/MPI, and PACX-MPI) in order to build the best MPI library available. A completely new MPI-2 compliant implementation, Open MPI offers advantages for system and software vendors, application developers and computer science researchers."

The implementations above have migrated into the OpenMPI project, which will make this an important implementation for the future. It offers full MPI-2 standards conformance. One of the things they have already created, is yet another way of starting an MPI job. It is called OpenRTE, which stands for The Open Run-Time Environment [52]. This is a spin-off from the Open MPI initiative. Its overarching goals are to create a portable, high-performance run-time environment for both serial and parallel applications in a wide variety of hardware and software environments, and to transparently support both single and multiple applications operating across a variety of environments.

OpenRTE supports a number of "launchers", listed below:

- rsh / ssh
- Recent versions of BProc (e.g., Clustermatic)
- PBS Pro, Torque, and Open PBS (the TM system)
- Yod (Red Storm)
- POE
- SLURM
- XGrid

2.4.5 OpenMPI on the WLCG

Since OpenMPI includes OpenRTE, and OpenRTE provides support for Torque, it is an excellent candidate for use on the WLCG. The `orterun` command is used to start an MPI application. It automatically recognizes if it is running under Torque/PBS and then uses the TM interface.

Just as for the other implementations, Charles Loomis has provided RPM's for OpenMPI as well. Unfortunately they are a little outdated, since the RPM's are for version 1.0.1, while 1.1 is the current release. It installs nicely into a Worker Node, because it is installed in `/opt/openmpi-1.0.1`. A sample wrapper script can be found in Appendix E.4.

2.4.6 Mpiexec

Mpiexec is not an MPI implementation, but a helper program, to ease integration with Torque/PBS. Is created by OSC, the Ohio Supercomputer Center. To quote their website [54]:

> "Mpiexec is a replacement program for the script mpirun, which is part of the mpich package. It is used to initialize a parallel job from within a PBS batch or interactive environment. Mpiexec uses the task manager library of PBS to spawn copies of the executable on the nodes in a PBS allocation."

The following MPI implementations are supported by Mpiexec:

MPICH/p4 Basic implementation of MPI over ethernet devices.

MPICH/gm MPI using GM or MX message-passing software on Myrinet hardware.

MVAPICH An MPI implementation for InfiniBand networks, based on MPICH.

MPICH/rai An MPI implementation for the IB-like network on the Cray XD1 products.

MPICH/shmem MPI using only the local memory bus on a single-node multi-processor machine.

MPICH2/PMI Rewrite of the venerable MPICH, with a different job launch interface (called PMI) that is intended to be common across all interconnects. Start mpiexec with –comm=pmi to specify this MPI.

MVAPICH2 An InfiniBand implementation based on MPICH2. Uses the same PMI interface for startup (–comm=pmi).

Intel MPI Based on mpich2 and uses the same PMI interface for startup.

EMP Experimental programmable gigabit ethernet MPI implementation.

none A way to start up many processes which are not related to each other. Handy for startup and shutdown activities or multiple single-processor tasks in a single script.

2.4.7 Mpiexec on the WLCG

Mpiexec is already deployed on all Worker Nodes (WN's), but in a rather old version, 0.77. This version had no support for MPICH2 yet. The current stable version is 0.80 and does have support for MPICH as well as MPICH2 as listed above. An RPM is provided by Charles Loomis that installs in `/opt/mpiexec-0.80/`. This means the path to `mpiexec` has to be provided for the user.

2.5 Modifications for deployment of MPI

TBD

2.6 Conclusion on MPI on the WLCG

TBD

Chapter 3

YAIM

Yet Another Installation Method, or *YAIM*, is a tool for installation and configuration of grid middleware. It was originally created by Louis Poncet[1]. It started as a straightforward, simple and easy to use tool, to help system administrators configure their machines. Before YAIM was created, system administrators had to work through a 300 page installation and configuration manual. This manual still exists, but now more as a reference guide. The installation and configuration manual is now reduced to some 30 pages.

3.1 Structure

YAIM is entirely written in Bash, a UNIX scripting language, except for a few scripts in Python that are used to create gLite configuration files. The gLite middleware itself contains Python scripts for configuration, which can be controlled and run from YAIM, but not after the configuration files have been created.

YAIM is a very modular system, the components are configuration files, scripts, functions and utilities.

Configuration files are what guide the configuration of the machine.

- The `site-info.def` file looks like a shell script, consisting only of attribute-value assignments. It is not to be regarded as a shell script, but solely as a configuration file. Functionality beyond setting a variable should be done in function files. In principle, one `site-info.def` should suffice for an entire site.
- The `node-info.def` file defines what steps need to be taken to configure a node to take on a certain role. This means that for every role, there is a list of functions that need to be called.

Scripts are the glue of YAIM.

- The `install_node` script manages the installation of grid middleware components. This comes down to installing a meta-package, which is an RPM file named after a grid middleware node type, that has dependencies on all software necessary to install this node type. Node types can be CE, WN, Torque-server, etcetera. It is possible for a node to have multiple types.
- The `configure_node` script is run after the `install_node` script and takes care of configuring the components based on the configuration files listed above. It does this by scanning the command line for arguments that match node types. These node types are then looked up in `node-info.def` and the resulting functions are run.

[1] Who happens to be our supervisor at CERN

Functions are the basic elements of configuration. For every element that needs to be configured, a file is created named after the function, and it only contains this function[2].

Sometimes a site needs to override the default version of a function. This is possible by creating a file named after the function in the `local` directory below the `functions` directory.

Utilities are small scripts that contain a routine that may be called by various functions.

3.2 Scope

As mentioned above, YAIM originated as a solution to help reduce the burden on system administrators. It was not built as a full featured configuration management system for large sites. Actually such a system has been built, as a result of Work Package 4 of EDG. It's task definition was [55]:

> "WP4 Installation system provides a mean to install operating system and applications through the network, to configure system parameters and thereby to apply site policies. It makes available all the necessary applications and user environments on the node. It runs automatically or on request to examine the system and eventually to bring it to the externally defined state stored in the 'configuration database'."

The resulting system from WP4 is Quattor [56]. This is in fact in use at a number of the bigger WLCG sites. But the scope of Quattor (and WP4), was installation and configuration of the Operating System. Not grid middleware.

There is a working group, called the Quattor Working Group (QWG), whose aim is to install, configure and maintain the grid middleware using Quattor [57]. This effectively extends the use of Quattor so that YAIM is no longer required[3].

3.3 Evaluation

TBD

underscores in vo namen

3.4 Conclusion on YAIM

YAIM does what it was built for.

[2] A helper function may be defined in a function file, but only if it is not possible to make it so generic that it should be a utility

[3] Or perhaps only for the QWG to do reverse engineering

Bibliography

[1] CERN: European Organization for Nuclear Research, http://www.cern.ch

[2] NIKHEF: Dutch Institute for High Energy and Nuclear Physics, http://www.nikhef.nl

[3] LCG Project Overview, http://lcg.web.cern.ch/LCG/overview.html

[4] Shiers, Sheets of SC4/WLCG Workshop, 2006, http://indico.cern.ch/materialDisplay.py?materialId=slides&confId=1148

[5] Gropp, William and Lusk, Ewing: *"PVM and MPI are completely different"*, 1997, http://citeseer.ist.psu.edu/573977.html

[6] MPI Forum: Message Passing Interface (MPI) Forum homepage, 1998, http://www.mpi-forum.org/docs/docs.html

[7] Fagg, Graham and London, Kevin: *"MPI Inter-connection and Control"*, 1998, http://citeseer.ist.psu.edu/400213.html

[8] Foster, Ian: *"What is the Grid? A Three Point Checklist."*, 2002, http://www-fp.mcs.anl.gov/∼foster/Articles/WhatIsTheGrid.pdf

[9] Foster, Ian: *"Service-Oriented Science."*, 2005, http://www.sciencemag.org/cgi/content/short/308/5723/814

[10] Papazoglou, M. and Georgakopoulos, D.: *"Service-oriented computing: Introduction"*, 2003, http://portal.acm.org/citation.cfm?doid=944217.944233

[11] Gropp, William and Lusk, Ewing: *"Goals Guiding Design: PVM and MPI"*, 2002, http://citeseer.ist.psu.edu/568858.html

[12] Trustees of Indiana University: *"MPI Implementation List"*, 2005, http://www.lam-mpi.org/mpi/implementations/

[13] NCSA / PACS Training Group: *"Introduction to MPI"*, 2001, http://webct.ncsa.uiuc.edu:8900/public/MPI/

[14] MPI-Forum: *"MPI: A Message-Passing Interface Standard"*, 1995, http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html

[15] EGEE TCG: *"Message Passing (MPI) WG"*, http://egee-intranet.web.cern.ch/egee-intranet/NA1/TCG/wgs/mpi.htm

[16] EGEE MPI Working Group: Minutes of the 2006-02-23 meeting, http://agenda.cern.ch/fullAgenda.php?ida=a061204

[17] Johnson, Mike: *"Superscalar Microprocessor Design"*, Prentice Hall, 1991

[18] IETF (S. Bradner): *"RFC 2119: Key words for use in RFCs to Indicate Requirement Level"*, 1997, http://www.ietf.org/rfc/rfc2119.txt

[19] EGEE JRA-1: *"Job Description Language Attributes Specification (submission through Network Server)"*, 2006, https://edms.cern.ch/document/555796/1

[20] EGEE JRA-1: *"Job Description Language Attributes Specification (submission through WM-Proxy)"*, 2006, https://edms.cern.ch/document/590869/1

[21] Gropp, William and Lusk, Ewing: *"Dynamic Process Management in an MPI Setting"*, 1995, http://citeseer.ist.psu.edu/gropp95dynamic.html

[22] Barret, Brian et al: *"Integration of the LAM/MPI environment and the PBS scheduling system"*, 2003, http://citeseer.ist.psu.edu/633226.html

[23] Saphir, William et al: *"Job Management Requirements for NAS Parallel Systems and Clusters"*, 1995, http://citeseer.ist.psu.edu/saphir95job.html

[24] Sistare, Steve et al: *"An Architecture for Integrated Resource Management of MPI Jobs"*, 2002, http://doi.ieeecomputersociety.org/10.1109/CLUSTR.2002.1137769

[25] Northern Illinois University: *"MPICH-G2"*, 2005, http://www3.niu.edu/mpi/

[26] Humphrey, Marty et al: *"Legion MPI: High Performance in Secure, Cross-Site, Cross-Architecture MPI Applications"*, 2001, http://legion.virginia.edu/presentations/LegionMPI-NAVO-Jun2001/

[27] Natrajan, Anand et al: *"Capacity and Capability Computing using Legion"*, 2001, http://citeseer.ist.psu.edu/natrajan01capacity.html

[28] Dong, Suchuan et al: *"Cross-Site Computations on the TeraGrid"*, 2005, Computing in Science and Engineering 7(5):14 – 23, ISBN: 1521-9615

[29] Karonis, N. et al: *"MPICH-G2: A Grid-enabled implementation of the Message Passing Interface"*, 2003, Journal of Parallel and Distributed Computing, 63(5):551 – 563

[30] Open MPI: *"Development Mailing List Archives"*, 2006, http://www.open-mpi.org/community/lists/devel/2006/06/0939.php

[31] The PSCHED API Working Group: *"PSCHED: An API for Parallel Job/Resource Management"*, 1996, http://www.pbspro.com/docs/psched-api-report.ps

[32] Squyres, Jeffrey et al: *"The interoperable message passing interface (IMPI) extensions to LAM/MPI"*, 2000, http://citeseer.ist.psu.edu/squyres00interoperable.html

[33] Skjellum, Anthony et al: *"Interoperability of Message-Passing Interface (MPI) Implementations: A Position Paper"*, 1997, http://citeseer.ist.psu.edu/68559.html

[34] HP: *"HP MPI User's Guide - Chapter 3"*, 2003, http://docs.hp.com/en/B6060-96013/ch03s03.html

[35] GridMPI: *"GridMPI version 1.1"*, 2006, http://www.gridmpi.org/gridmpi-1-1/

[36] Lastovetsky, Alexey et al: *"HeteroMPI: Towards a message-passing library for heterogeneous networks of computers"*, 2006, Journal of Parallel and Distributed Computing, 66(6):197 – 220

[37] Peisert, Sean Philip: *"A Programming Model for Automated Decomposition on Heterogeneous Clusters of Multiprocessors"*

[38] Mohamed, H.H. et al: *"The Design and Implementation of the KOALA Co-Allocating Grid Scheduler"*, 2005, http://www.st.ewi.tudelft.nl/koala/papers/egc-final00.ps

[39] Matsuda, Motohiko et al: *"Evaluation of MPI Implementations on Grid-connected Clusters using an Emulated WAN Environment"*, 2003, Proceedings of the 3st International Symposium on Cluster Computing and the Grid (page 10), ISBN:0-7695-1919-9

[40] IETF (R. Braden): *"RFC 1122: Requirements for Internet Hosts – Communication Layers"*, 1989, http://www.ietf.org/rfc/rfc1122.txt

[41] Matsuda, Motohiko et al: *"TCP Adaptation for MPI on Long-and-Fat Networks"*, 2005, Proceedings of the IEEE International Conference on Cluster Computing (Cluster 2005)

[42] Matsuda, Motohiko et al: *"The Design and Implementation of an Asynchronous Communication Mechanism for the MPI Communication Model"*, 2004, Proceedings of the IEEE International Conference on Cluster Computing (Cluster 2004)

[43] Kamal, Humaira et al: *"SCTP versus TCP for MPI"*, 2005, Proceedings of the ACM/IEEE SC 2005 Conference (SC'05), p.30 – ?

[44] CERN: *"LCG Computing Grid - Technical Design Report v1.04"*, 2005, http://lcg.web.cern.ch/LCG/tdr/LCG_TDR_v1_04.pdf

[45] JRA-1: *"Job Description Language - Attributes Specification v0.8"*, 2006, https://edms.cern.ch/file/555796/1/EGEE-JRA1-TEC-555796-JDL-Attributes-v0-8.pdf

[46] Scientific Linux, website, https://www.scientificlinux.org/

[47] Laboratoire de lAcclrateur Linaire (LAL), website, http://www.lal.in2p3.fr/

[48] Institut national de physique nuclaire et de physique des particules (IN2P3), website, http://www.in2p3.fr/

[49] Index of MPI Packages, http://quattor.web.lal.in2p3.fr/packages/mpi/

[50] Argonne National Laborotary, website, http://www.anl.gov/

[51] OpenMPI, A High Performance Message Passing Library, website, http://www.open-mpi.org/

[52] The Open Run-Time Environment (ORTE) Project, website, http://www.open-rte.org/

[53] LAM/MPI Parallel Computing, website, http://www.lam-mpi.org/

[54] Ohio Supercomputer Center, website, http://www.osc.edu/∼pw/mpiexec/index.php

[55] Polok and Iven: *"WP4 Task definition: Installation"*, 2001, http://iven.home.cern.ch/iven/wp4/wp4-inst-task.pdf

[56] QUATTOR: QUattor is an Administration ToolkiT for Optimizing Resources, website, http://quattor.web.cern.ch/quattor/

[57] Quattor Working Group, https://svn.lal.in2p3.fr/LCG/QWG/web/index.html

[58] Creative Commons: Creative Commons Attribution 2.5 license, www.creativecommons.org

[59] Open Source Initiative, The BSD License, www.opensource.org

Appendix A

The BSD license for this project

Appendix B

Requirements

In Q1/2006, a new working group was formed within EGEE Technical Coordination Group (EGEE-TCG), specifically focussing on MPI [15]. In their early meetings, preliminary end-user and (grid) site requirements were elicited. In the next subsections, the explicit requirements are listed and tagged for referencing purposes (i.e. traceability) and prioritized **for our own purposes** using the conventional {MUST,SHOULD,MAY} requirement levels described in RFC 2119 [18]. The prioritization has been refined during a phone conference with Charles Loomis, who is leader of the EGEE working group.

B.1 User requirements

The end-user requirements are as follows [16]:

u1 MUST *"Users need to prepare the job before the MPI executable is launched. This typically involves compilation of the executable and preparation of the job inputs."*

u2 MUST *"Some jobs require a shared directory. During the execution of the job, some of the files in the shared directory are updated. These changes need to be visible on the child nodes."*

u3 MUST *"Multiple versions of MPI need to be supported. There is a mix of MPI-1 and MPI-2 jobs which need to be executed. Moreover, some users prefer a particular implementation. We should look at MPICH-1, MPICH-2, and LAM at a minimum."*

u4 MAY *"At least the computational chemistry application will need to have cross-site MPI available. This means looking at MPICH-G2 and other similar solutions. The experiences from CrossGrid will be helpful here."*

u5 MAY *"Many of the MPI codes are compiled with commercial compilers. We need to see how this can be accommodated on the production service."*

u6 SHOULD *"Some grid-enabled applications call grid services from the child processes. This means that these processes need access to the user proxy. This access is usually not possible because the proxy is put into a local space (/tmp). Currently the applications handle the distribution of the proxies directly."*

u7 MAY *"Some applications want to have detailed control over the distribution of jobs, specifying both the number of nodes and number of CPUs on each node."*

B.2 Site requirements

The site requirements are as follows [16]:

s1 MAY *"Need to ensure that MPI jobs are compatible with all of the job managers (lcgpbs, pbs, etc.)."*

s2 MUST *"The details of the 'shared home directory' requirement needs to be better understood. First, is it necessary that this be the home directory or can another directory be provided? Second, is this required for all jobs; could some jobs survive without? Third, if jobs can survive without, what files need to be distributed automatically to the child nodes? The executable is the minimum, but proxies, etc. may be needed."*

s3 MAY *"It looks like most sites supporting MPI on the production service will support a mix of MPI and non-MPI jobs. Efficient scheduling can only be done in this situation if job backfilling is possible. This means that the local scheduler *must* have access to the job requirements. This requirement has come up in other contexts. It is critical for MPI jobs to avoid inefficient use of the fabric or job starvation. For the case of starvation, Fokke [Dijkstra] pointed out that the new ERT algorithm may help avoid some of the bad scheduling decisions."*

s4 MAY *"Need to look at efficient scheduling of the MPI jobs at the grid-level at well. For the CrossGrid modified resource broker some additional information was needed, such as the number of CPUs really free, number of CPUs available for MPI, and the like. May also need schema changes for indicating whether a shared home directory is available."*

s5 MAY *"CrossGrid also found that there were some problems with the standard LRMS information providers. For example for PBS, it would advertise a CPU as free even if it was loaded and could not be scheduled. Need to look in detail to ensure that the information is correct."*

s6 SHOULD *"Need to avoid hardcoding of things at the level of the resource broker. This makes it extremely difficult to customize MPI support at the site. For example, there is a desire for site running Torque/PBS to use mpiexec which permits correct accounting and control of the child processes. The job wrapper should not assume that this is mpirun, nor call it automatically at the start of the job."*

s7 SHOULD *"A variety of batch systems need to be supported. At the meeting there were people wanting to have SGE and Torque working with MPI. There are also many sites running LSF."*

s8 SHOULD *"Having a standard user environment came up in several of the points above. For example, it might be possible to provide an environment variable which indicates the location of a shared disk (not necessarily the home area). Also when supporting many different MPI implementations, there will be a need to indicate the locations of those implementations. Environment variables may also be a natural way of implementing site customizations, e.g. indicating the executable to start MPI."*

s9 MAY *"If cross-site MPI is supported, then the IP requirements for doing so must be determined."*

Appendix C

Changes to YAIM

C.1 functions/config_mpi

We have added a file to YAIM, namely `function/config_mpi`:

```
#
# CONFIG_MPI
#
# Description: MPI-related configuration
#
# History:
#     2006-06-21, mkoot: File created
#     2006-06-23, mkoot: Replaced "rpm -i" with "rpm -uvh"
#                        Added $MPI_MPIEXEC
#
# TODO ! TODO ! TODO: replace hardcoded "rpm -i" by "apt-get install"
#

config_mpi() {

cat <<EOF > /etc/profile.d/mpi.sh
#
# MPI.SH
#
# Description (short):
# --
# This script is generated by YAIM when running "configure_node <foobar.def> glite-WN".
# It should be executed as a prelude to an MPI job and sets MPI environment variables.
#
# Description (not quite that short):
# --
# The location of the MPI implementation is found by querying if the RPM is installed,
# searching for the mpicc command in the file list, extracting the path from it and
# testing if it exists in real life. You are free to use a fixed value for the
# $MPI_<foobar>_PATH and $MPI_<foobar>_VERSION, however, and arbitrary MPI implementations
# may be added (inform your users on the variable names). REMEMBER that you may want
# to have the CE advertise your custom MPI, and if so, the changes you make here
# need to be reflected in the CEs GlueSoftwareRunTimeEnvironment!
#

find_dir() {
        dir=‘rpm -ql $1 | awk -F/bin/mpicc ’/mpicc$/ {print $1}’‘
        if [ "x$dir" != "x" ]
        then
```

```
                if [ -d $dir -a -f $dir/bin/mpicc ]
                then
                        echo $dir
                fi
        fi
}

find_version() {
        version=`rpm -q $1 | grep -v "is not installed" | awk -F- '{print $2}'`
        if [ "x$version" != "x" ]
        then
                echo $version
        fi
}
EOF

#----
# Handle MPICH
#----
if [ "x$ENABLE_MPI_MPICH" == "xyes" ]
then
  if [ "x$ENABLE_MPI_MPICH_PATH" != "x" ]
  then
    echo "MPI_MPICH_PATH='$ENABLE_MPI_MPICH_PATH'">> /etc/profile.d/mpi.sh
    echo "MPI_MPICH_VERSION='$ENABLE_MPI_MPICH_VERSION'">> /etc/profile.d/mpi.sh
  else
    rpm -uvh "http://quattor.web.lal.in2p3.fr/packages/mpi/mpich-1.2.7p1-1.sl3.cl.1.i386.rpm"
    echo 'MPI_MPICH_PATH=`find_dir mpich`'>> /etc/profile.d/mpi.sh
    echo 'MPI_MPICH_VERSION=`find_version mpich`'>> /etc/profile.d/mpi.sh
  fi
fi

#----
# Handle MPICH2
#----
if [ "x$ENABLE_MPI_MPICH2" == "xyes" ]
then
  if [ "x$ENABLE_MPI_MPICH2_PATH" != "x" ]
  then
    echo "MPI_MPICH2_PATH='$ENABLE_MPI_MPICH2_PATH'">> /etc/profile.d/mpi.sh
    echo "MPI_MPICH2_VERSION='$ENABLE_MPI_MPICH2_VERSION'">> /etc/profile.d/mpi.sh
  else
    rpm -i "http://quattor.web.lal.in2p3.fr/packages/mpi/mpich2-1.0.3-1.sl3.cl.1.i386.rpm"
    echo 'MPI_MPICH2_PATH=`find_dir mpich2`'>> /etc/profile.d/mpi.sh
    echo 'MPI_MPICH2_VERSION=`find_version mpich2`'>> /etc/profile.d/mpi.sh
  fi
fi

#----
# Handle OpenMPI
#----
if [ "x$ENABLE_MPI_OPENMPI" == "xyes" ]
then
  if [ "x$ENABLE_MPI_OPENMPI_PATH" != "x" ]
  then
    echo "MPI_OPENMPI_PATH='$ENABLE_MPI_OPENMPI_PATH'">> /etc/profile.d/mpi.sh
    echo "MPI_OPENMPI_VERSION='$ENABLE_MPI_OPENMPI_VERSION'">> /etc/profile.d/mpi.sh
  else
    rpm -uvh "http://quattor.web.lal.in2p3.fr/packages/mpi/openmpi-1.0.1-1.sl3.cl.1.i386.rpm"
```

```
    echo 'MPI_OPENMPI_PATH=`find_dir openmpi`'>> /etc/profile.d/mpi.sh
    echo 'MPI_OPENMPI_VERSION=`find_version openmpi`'>> /etc/profile.d/mpi.sh
  fi
fi

#----
# Handle LAMv6
#----
if [ "x$ENABLE_MPI_LAM_6" == "xyes" ]
then
  if [ "x$ENABLE_MPI_LAM_6_PATH" != "x" ]
  then
    echo "MPI_LAM_6_PATH='$ENABLE_MPI_LAM_6_PATH'">> /etc/profile.d/mpi.sh
    echo "MPI_LAM_6_VERSION='$ENABLE_MPI_LAM_6_VERSION'">> /etc/profile.d/mpi.sh
  else
    # TBD: rpm -uvh SOMETHING?
    echo "RP2-DEBUG: all is OK, but there isn't a .rpm for LAM-6 yet so it won't be usable..."
    echo 'MPI_LAM_6_PATH=`find_dir lam-6*`'>> /etc/profile.d/mpi.sh
    echo 'MPI_LAM_6_VERSION=`find_version lam-6*`'>> /etc/profile.d/mpi.sh
  fi
fi

#----
# Handle LAMv7
#----
if [ "x$ENABLE_MPI_LAM_7" == "xyes" ]
then
  if [ "x$ENABLE_MPI_LAM_7_PATH" != "x" ]
  then
    echo "MPI_LAM_7_PATH='$ENABLE_MPI_LAM_7_PATH'">> /etc/profile.d/mpi.sh
    echo "MPI_LAM_7_VERSION='$ENABLE_MPI_LAM_7_VERSION'">> /etc/profile.d/mpi.sh
  else
    # TBD: rpm -uvh SOMETHING?
    echo "RP2-DEBUG: all is OK, but there isn't a .rpm for LAM-6 yet so it won't be usable..."
    echo 'MPI_LAM_7_PATH=`find_dir lam-7*`'>> /etc/profile.d/mpi.sh
    echo 'MPI_LAM_7_VERSION=`find_version lam-7*`'>> /etc/profile.d/mpi.sh
  fi
fi

#----
# Handle shared home and SSH host-based authentication parameters
#----
echo "MPI_PATH_TO_SHARED_HOME='$ENABLE_MPI_SHARED_HOME_PATH'">> /etc/profile.d/mpi.sh
echo "MPI_SSH_HOST_BASED_AUTH='$ENABLE_MPI_SSH_HOST_BASED_AUTH'">> /etc/profile.d/mpi.sh
echo 'MPI_MPIEXEC=`find_dir mpiexec`/mpiexec'>> /etc/profile.d/mpi.sh

#----
# Export
#----
echo "export MPI_MPICH_PATH MPI_MPICH_VERSION MPI_MPICH2_PATH MPI_MPICH2_VERSION MPI_OPENMPI_PATH MPI_OPENMPI_VE

   return 0
}
```

C.2 examples/site-info.def

We have added variables to the `site-info.def` file for specifying MPI-related parameters:

```
#----------------------------------
```

```
# MPI-related configuration:
#----------------------------------
# Several MPI implementations are available in the package repository.
# If you do NOT want an implementation to be installed, 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: 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.

ENABLE_MPI_MPICH="yes"
ENABLE_MPI_MPICH2="yes"
ENABLE_MPI_OPENMPI="yes"
ENABLE_MPI_LAM_6="no"
ENABLE_MPI_LAM_7="no"

#---
# Example for using an already installed version of MPI
# (this will prevent YAIM from installing MPICH through apt-get)
#---
#ENABLE_MPI_MPICH_PATH="/usr"
#ENABLE_MPI_MPICH_VERSION="1.2.6"
#---

MPI_CE_RUNTIMEENV=""
if [ "x$ENABLE_MPI_MPICH" = "xyes" ]
then
  MPI_CE_RUNTIMEENV="$MPI_CE_RUNTIMEENV MPICH"
fi

if [ "x$ENABLE_MPI_MPICH2" = "xyes" ]
then
  MPI_CE_RUNTIMEENV="$MPI_CE_RUNTIMEENV MPICH2"
fi

if [ "x$ENABLE_MPI_OPENMPI" = "xyes" ]
then
  MPI_CE_RUNTIMEENV="$MPI_CE_RUNTIMEENV OPENMPI"
fi

if [ "x$ENABLE_MPI_LAM_6" = "xyes" ]
then
  MPI_CE_RUNTIMEENV="$MPI_CE_RUNTIMEENV LAM_6"
fi

if [ "x$ENABLE_MPI_LAM_7" = "xyes" ]
then
  MPI_CE_RUNTIMEENV="$MPI_CE_RUNTIMEENV LAM_7"
fi


#
# If you do NOT provide a shared home, set $ENABLE_MPI_SHARED_HOME to "no" (default).
# Else, set it to "yes" and assign its path to $ENABLE_MPI_SHARED_HOME_PATH="/shared").
#
ENABLE_MPI_SHARED_HOME="no"
```

```
ENABLE_MPI_SHARED_HOME_PATH=""

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

#---------------------------------
```

Accordingly, the $CE_RUNTIMEENV variable is appended with the specified MPI environment (in the same file; this will be advertised by the CE once it's running):

```
(...)
CE_RUNTIMEENV="
    LCG-2
    LCG-2_1_0
    LCG-2_1_1
    LCG-2_2_0
    LCG-2_3_0
    LCG-2_3_1
    LCG-2_4_0
    LCG-2_5_0
    LCG-2_6_0
    LCG-2_7_0
    GLITE-3_0_0
    R-GMA
    $MPI_CE_RUNTIMEENV"
```

C.3 scripts/node-info.def

We have modified `scripts/node-info.def` so that the `functions/config_mpi` script is called during WN and CE deployment:

```
(...)

# This is /opt/glite/yaim/scripts/node-info.def.
# "config_mpi" has been appended to the WN_FUNCTIONS list
WN_FUNCTIONS="${BASE1_FUNCTIONS}
....
config_mpi"

(...)
```

Appendix D

A Quick Guide to using MPI on the WLCG

The following sections contain listings of three files altogether comprising an example MPI job.

D.1 mpi-example.jdl

```
#
# MPI-EXAMPLE.JDL
# (requires "mpi-example.{c,sh}")
#
# YEP, THE JOBTYPE IS "NORMAL"! This prevents the RB/WMS from doing
# some unwanted wrapping (i.e. "template.mpi.<jms>.sh").
#
JobType = "NORMAL";
NodeNumber = 3;
Executable = "mpi-example.sh";
StdOutput = "mpi-example.out";
StdError = "mpi-example.err";
InputSandbox = {"mpi-example.sh","mpi-example.c"};
OutputSandbox = {"mpi-example.out","mpi-example.err","mpiexec.out"};
Requirements = other.GlueCEUniqueID == "lxb1929.cern.ch:2119/blah-pbs-dteam";
```

D.2 mpi-example.c

```
/*
 *  MPI-EXAMPLE.C
 * (requires "mpi-example.{jdl,sh}")
 *
 */

#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[])
{
  int numprocs;  /* Number of processors */
  int procnum;   /* Processor number */
  /* Initialize MPI */
  MPI_Init(&argc, &argv);
  /* Find this processor number */
  MPI_Comm_rank(MPI_COMM_WORLD, &procnum);
```

```
  /* Find the number of processors */
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  printf ("Hello world! from processor %d out of %d\n", procnum, numprocs);
  /* Shut down MPI */
  MPI_Finalize();
  return 0;
}
```

D.3 mpi-example.sh

```
#!/bin/sh
#
# MPI-EXAMPLE.SH
# (requires "mpi-example.{c,jdl}")
#

#-------------
# Select the MPI version by referring to one of the WN-provided
# environment variables:
#
# - $MPI_MPICH_PATH (for MPICH1, i.e. MPIv1)
# - $MPI_MPICH2_PATH (for MPICH2, i.e. MPIv2)
# - $MPI_OPENMPI_PATH (for OPENMPI, i.e. MPIv2)
# - $MPI_LAM_6_PATH (for LAMv6)
# - $MPI_LAM_7_PATH (for LAMv7)
# - $MPI_<vendor>_PATH (custom vendor MPI)
#-------------

echo "Testing MPICH"

if [ -z $MPI_MPICH_VERSION ]
then
        echo "MPICH, no such version available"
        exit
fi

PATH_TO_MPI=$MPI_MPICH_PATH


#-------------
# Prepare the environment
#-------------
export PATH="$PATH_TO_MPI/bin:$PATH"
export LD_LIBRARY_PATH="$PATH_TO_MPI/lib:$LD_LIBRARY_PATH"
export LIBRARY_PATH="$PATH_TO_MPI/lib:$LIBRARY_PATH"
export CPATH="$PATH_TO_MPI/include:$CPATH"

#-------------
# Compile
#-------------
# The binary to execute
EXE=$1

echo "**************************************************************************"
echo "Running on: $HOSTNAME"
echo "As:       " `whoami`
echo "**************************************************************************"
```

```
echo "************************************************************************"
echo "Compiling binary: $EXE"
echo mpicc -o ${EXE} ${EXE}.c
mpicc -o ${EXE} ${EXE}.c
echo "************************************"

if [ "x$PBS_NODEFILE" != "x" ] ; then
  echo "PBS Nodefile: $PBS_NODEFILE"
  HOST_NODEFILE=$PBS_NODEFILE
fi

if [ "x$LSB_HOSTS" != "x" ] ; then
  echo "LSF Hosts: $LSB_HOSTS"
  HOST_NODEFILE=`pwd`/lsf_nodefile.$$
  for host in ${LSB_HOSTS}
  do
    echo $host >> ${HOST_NODEFILE}
  done
fi

if [ "x${HOST_NODEFILE}" = "x" ]; then
  echo "No hosts file defined.  Exiting..."
  exit
fi


if [ "x${MPI_PATH_TO_SHARED_HOME}" != "x" ]
then
        if [ ! -d ${MPI_PATH_TO_SHARED_HOME} ]
        then
                echo "Shared home is set, but it is not accessible"
                exit 1
        else
                echo "Shared home exists and is ${MPI_PATH_TO_SHARED_HOME}"
        fi
else
        echo "No shared home is accessible"
        if [ "x${MPI_SSH_HOST_BASED_AUTH}" == "xyes" ]
        then
                echo "Using SSH to copy the files to all nodes"

                for i in `cat $HOST_NODEFILE | grep -v ${HOSTNAME}`; do
                        ssh ${i} mkdir -p `pwd`
                        scp -rp . "${i}:`pwd`"
                        # Copy the proxy certificate
                        scp -p ${X509_USER_PROXY} ${i}:/tmp
                done
        else
                echo "Uh oh, cannot use SSH to copy the files to all WN's"
                exit 1
        fi
fi


echo "************************************************************************"
HOST_NEEDED=`cat $HOST_NODEFILE | wc -l`
echo "Node count: $HOST_NEEDED"
echo "CPU count: $NRPROCS"
echo "Nodes in $HOST_NODEFILE: "
```

```
cat $HOST_NODEFILE
echo "**************************************************************************"


#-------------
# Execute
#-------------

echo "**************************************************************************"
echo "Executing $EXE with mpiexec"
chmod 755 $EXE
mpiexec -verbose ‘pwd‘/$EXE > mpiexec.out 2>&1
echo "**************************************************************************"

echo "**************************************************************************"
echo "Executing $EXE with mpirun"
chmod 755 $EXE
mpirun -np $NRPROCS -machinefile $HOST_NODEFILE ‘pwd‘/$EXE
echo "**************************************************************************"
```

Appendix E

Wrapper scripts

E.1 lam-6-wrapper.sh

```
# Script to start an MPI job under lam-6
# Richard de Jong @ CERN 2006-06-20

PATH=$MPI_LAM_6_PATH/bin:$PATH
LD_LIBRARY_PATH=$MPI_LAM_6_PATH/lib:$LD_LIBRARY_PATH
SHUTDOWN="yes"
EXE=$1

if [ "x${PBS_NODEFILE}" != "x" ] ; then
  echo "PBS Nodefile: ${PBS_NODEFILE}"
  HOST_NODEFILE=${PBS_NODEFILE}
fi

if [ "x${LSB_HOSTS}" != "x" ] ; then
  echo "LSF Hosts: ${LSB_HOSTS}"
  HOST_NODEFILE=`pwd`/lsf_nodefile.$$
  for host in ${LSB_HOSTS}
  do
    echo ${host} >> ${HOST_NODEFILE}
  done
fi

if [ "x${HOST_NODEFILE}" = "x" ]; then
  echo "No hosts file defined. Exiting..."
  exit
fi

# Check if there is already a running LAM environment set up:

if ( lamnodes > /dev/null 2>&1 ) then
  echo "lam is set up already"
else
  echo "lam not ready yet, booting lam environment using hosts from ${HOST_NODEFILE}"
  if ( lamboot ${HOST_NODEFILE} )
  then
    echo "lam booted succesfully"
  else
                echo "Error booting lam. Exiting..."
                exit 1
        fi
```

```
fi

# In case the lam environment had already been started, make sure we only use
# the hosts that are assigned to us by the LRMS. This provides at least a
# lightweigt coupling.

AVAILABLE_NODES=‘lamnodes | grep -f ${HOST_NODEFILE} |awk ’{print $1}’ | tr \
’\n’ ’ ’‘ echo "available nodes: ${AVAILABLE_NODES}"

MY_NODE_NUMBER=‘lamnodes h -c|awk ’{print $1}’‘
if [ "x${MY_NODE_NUMBER}" == "x" ] then
  echo "Localhost is not one of the running lam nodes."
  echo "Can not continue because only this host has the executable.  Exiting..."
  exit 1
fi

echo "compiling program ${EXE}"
mpicc -o $EXE $EXE.c
if [ ! $? -eq 0 ]; then
  echo "Error compiling program.  Exiting..."
  exit 1
fi

# Run the command on the available nodes, using -s helps to get the executable
# to all hosts
mpirun -s ${MY_NODE_NUMBER} ${AVAILABLE_NODES} ${EXE}

#close down the lam environment?
if [ "${SHUTDOWN}" == "yes" ] then
  lamhalt
fi
```

E.2 mpich2-wrapper.sh for non-Torque/PBS installations

This script sets up the mpd-ring because there is no support in MPICH2 for other LRMS's than Torque/PBS

```
#!/bin/bash

echo "Testing MPICH2"

if [ -z $MPI_MPICH2_VERSION ]
then
        echo "MPICH2, no such version available"
        exit
fi

# Set up the paths
PATH=$MPI_MPICH2_PATH/bin:$PATH
LD_LIBRARY_PATH=$MPI_MPICH2_PATH/lib:$LD_LIBRARY_PATH

# The binary to execute
EXE=$1

echo "****************************************************************************"
echo "Running on: $HOSTNAME"
echo "As:       " ‘whoami‘
echo "****************************************************************************"
```

```
echo "************************************************************************"
echo "Compiling binary: $EXE"
echo mpicc -o ${EXE} ${EXE}.c
mpicc -o ${EXE} ${EXE}.c
echo "************************************"


# Create the .mpd.conf file.
# This is required to boot the mpd daemons on all hosts
# $RANDOM creates a random integer between 0 and 32767.
echo "MPD_SECRETWORD=PASS$RANDOM" > ~/.mpd.conf
chmod 0600 ~/.mpd.conf

if [ "x$PBS_NODEFILE" != "x" ] ; then
        echo "PBS Nodefile: $PBS_NODEFILE"
        HOST_NODEFILE=$PBS_NODEFILE
fi

if [ "x$LSB_HOSTS" != "x" ] ; then
        echo "LSF Hosts: $LSB_HOSTS"
        HOST_NODEFILE=`pwd`/lsf_nodefile.$$
        for host in ${LSB_HOSTS}
        do
                echo $host >> ${HOST_NODEFILE}
        done
fi

if [ "x${HOST_NODEFILE}" = "x" ]; then
        echo "No hosts file defined.  Exiting..."
        exit
fi


if [ "x${MPI_PATH_TO_SHARED_HOME}" != "x" ]
then
        if [ ! -d ${MPI_PATH_TO_SHARED_HOME} ]
        then
                echo "Shared home is set, but it is not accessible"
                exit 1
        else
                echo "Shared home exists and is ${MPI_PATH_TO_SHARED_HOME}"
        fi
else
        echo "No shared home is accessible"
        if [ "x${MPI_SSH_HOST_BASED_AUTH}" == "xyes" ]
        then
                echo "Using SSH to copy the files to all nodes"

                for i in `cat $HOST_NODEFILE | grep -v ${HOSTNAME}`; do
                        ssh ${i} mkdir -p `pwd`
                        scp -rp . "${i}:`pwd`"
                        # Copy the proxy certificate
                        scp -p ${X509_USER_PROXY} ${i}:/tmp
                        # Copy the MPD secret file
                        scp -p ~/.mpd.conf ${i}:

                done
        else
                echo "Uh oh, cannot use SSH to copy the files to all WN's"
```

```
                exit 1
        fi
fi

HOST_NEEDED=`cat ${HOST_NODEFILE} | wc -l`
CPU_PER_HOST=$((${NRPROCS}/${HOST_NEEDED}))
if [[ ${CPU_PER_HOST} -eq 0 ]]
then
        echo "$NRPROCS and $HOST_NEEDED incompatible,"
        echo "probably less CPU's specified than HOSTS"
        exit 1
fi

# Boot the MPD ring
mpdboot --ncpus=${CPU_PER_HOST} -n ${HOST_NEEDED} -f ${HOST_NODEFILE}


echo "************************************************************************"
echo "Testing MPD ring with mpdtrace
mpdtrace
echo "Testing MPD ring with mpdringtest
mpdringtest
echo "************************************************************************"

echo "************************************************************************"
echo "Executing $EXE with mpiexec"
mpiexec -n ${NRPROCS} `pwd`/${EXE}
echo "************************************************************************"

# Stop the MPD daemons
mpdallexit
```

E.3 mpich2-wrapper.sh for Torque/PBS installations

This wrapper script does not use the mpd commands, because the mpiexec that integrates with TM is used.

```
#!/bin/bash

echo "Testing MPICH2"

if [ -z $MPI_MPICH2_VERSION ]
then
        echo "MPICH2, no such version available"
        exit
fi

# Set up the paths
PATH=$MPI_MPICH2_PATH/bin:$PATH
LD_LIBRARY_PATH=$MPI_MPICH2_PATH/lib:$LD_LIBRARY_PATH

# The binary to execute
EXE=$1

echo "************************************************************************"
echo "Running on: $HOSTNAME"
echo "As:       " `whoami`
echo "************************************************************************"
```

```
echo "************************************************************************"
echo "Compiling binary: $EXE"
echo mpicc -o ${EXE} ${EXE}.c
mpicc -o ${EXE} ${EXE}.c
echo "************************************"


if [ "x$PBS_NODEFILE" != "x" ] ; then
        echo "PBS Nodefile: $PBS_NODEFILE"
        HOST_NODEFILE=$PBS_NODEFILE
fi

if [ "x$LSB_HOSTS" != "x" ] ; then
        echo "LSF Hosts: $LSB_HOSTS"
        HOST_NODEFILE=`pwd`/lsf_nodefile.$$
        for host in ${LSB_HOSTS}
        do
                echo $host >> ${HOST_NODEFILE}
        done
fi

if [ "x${HOST_NODEFILE}" = "x" ]; then
        echo "No hosts file defined.  Exiting..."
        exit
fi


if [ "x${MPI_PATH_TO_SHARED_HOME}" != "x" ]
then
        if [ ! -d ${MPI_PATH_TO_SHARED_HOME} ]
        then
                echo "Shared home is set, but it is not accessible"
                exit 1
        else
                echo "Shared home exists and is ${MPI_PATH_TO_SHARED_HOME}"
        fi
else
        echo "No shared home is accessible"
        if [ "x${MPI_SSH_HOST_BASED_AUTH}" == "xyes" ]
        then
                echo "Using SSH to copy the files to all nodes"

                for i in `cat $HOST_NODEFILE | grep -v ${HOSTNAME}`; do
                        ssh ${i} mkdir -p `pwd`
                        scp -rp . "${i}:`pwd`"
                        # Copy the proxy certificate
                        scp -p ${X509_USER_PROXY} ${i}:/tmp
                done
        else
                echo "Uh oh, cannot use SSH to copy the files to all WN's"
                exit 1
        fi
fi

echo "Executing $EXE with $MPI_MPIEXEC from the mpiexec package"
$MPI_MPIEXEC `pwd`/${EXE}
```

E.4 openmpi-wrapper.sh

```
#!/bin/sh

echo "Testing OpenMPI"

if [ -z $MPI_OPENMPI_VERSION ]
then
        echo "OPENMPI, no such version available"
        exit
fi

# Set up the paths
PATH=$MPI_OPENMPI_PATH/bin:$PATH
LD_LIBRARY_PATH=$MPI_OPENMPI_PATH/lib:$LD_LIBRARY_PATH

# The binary to execute
EXE=$1

echo "*****************************************************************"
echo "Running on: $HOSTNAME"
echo "As:       " `whoami`
echo "*****************************************************************"

echo "*****************************************************************"
echo "Compiling binary: $EXE"
echo mpicc -o ${EXE} ${EXE}.c
mpicc -o ${EXE} ${EXE}.c
echo "***********************************"

if [ "x$PBS_NODEFILE" != "x" ] ; then
        echo "PBS Nodefile: $PBS_NODEFILE"
        HOST_NODEFILE=$PBS_NODEFILE
fi

if [ "x$LSB_HOSTS" != "x" ] ; then
        echo "LSF Hosts: $LSB_HOSTS"
        HOST_NODEFILE=`pwd`/lsf_nodefile.$$
        for host in ${LSB_HOSTS}
        do
                echo $host >> ${HOST_NODEFILE}
        done
fi

if [ "x${HOST_NODEFILE}" = "x" ]; then
        echo "No hosts file defined.  Exiting..."
        exit
fi


if [ "x${MPI_PATH_TO_SHARED_HOME}" != "x" ]
then
        if [ ! -d ${MPI_PATH_TO_SHARED_HOME} ]
        then
                echo "Shared home is set, but it is not accessible"
                exit 1
        else
                echo "Shared home exists and is ${MPI_PATH_TO_SHARED_HOME}"
        fi
```

```
else
        echo "No shared home is accessible"
        if [ "x${MPI_SSH_HOST_BASED_AUTH}" == "xyes" ]
        then
                echo "Using SSH to copy the files to all nodes"

                for i in `cat $HOST_NODEFILE | grep -v ${HOSTNAME}`; do
                        ssh ${i} mkdir -p `pwd`
                        scp -rp . "${i}:`pwd`"
                        # Copy the proxy certificate
                        scp -p ${X509_USER_PROXY} ${i}:/tmp
                done
        else
                echo "Uh oh, cannot use SSH to copy the files to all WN's"
                exit 1
        fi
fi


echo "************************************************************************"
HOST_NEEDED=`cat $HOST_NODEFILE | wc -l`
echo "Node count: $HOST_NEEDED"
echo "CPU count: $NRPROCS"
echo "Nodes in $HOST_NODEFILE: "
cat $HOST_NODEFILE
echo "************************************************************************"

echo "************************************************************************"
echo "Executing $EXE with orterun"
chmod 755 $EXE
orterun --prefix $MPI_OPENMPI_PATH -np $NRPROCS -machinefile $HOST_NODEFILE `pwd`/$EXE
echo "************************************************************************"
```