Computing and Informatics, Vol. 27, 2008, 213–222

MPI SUPPORT ON THE GRID

Kiril Dichev, Sven Stork, Rainer Keller

High Performance Computing Center
University of Stuttgart
Nobelstrasse 19
70569 Stuttgart, Germany
e-mail: {dichev, keller}@hlrs.de

Enol Fernández

Universidad Autonoma de Barcelona
Barcelona, Spain
e-mail: enol@aomail.uab.es

Keywords: MPI, grid, metacomputing, workload management system

Abstract. Grids as infrastructures offer access to computing, storage and other resources in a transparent way. The user does not have to be aware where and how the job is being executed. Grid clusters in particular are an interesting target for running computation-intensive calculations. Running MPI-parallel applications on such clusters is a logical approach that is of interest to both computer scientists and to engineers. This paper gives an overview of the issues connected to running MPI applications on a heterogenous Grid consisting of different clusters located at different sites within the Int.EU.Grid project. The role of a workload management system (WMS) for such a scenario, as well as important modifications that need to be made to a WMS oriented towards sequential batch jobs for better support of MPI applications and tools are discussed. In order to facilitate the adoption of MPI-parallel applications on heterogeneous Grids, the application developer should be made aware of performance problems, as well as MPI-standard issues within its code. Therefore tools for these issues are also supported within Int.EU.Grid. Also, the special case of running MPI applications on different clusters simultaneously as a more Grid-oriented computational approach is described.

1 INTRODUCTION

Computational Grid infrastructures offer vast amounts of compute power for applications. In order to make best usage of the clusters within a given Grid, a performant MPI implementation and MPI tools have to be provided on the clusters, which may be equipped each with different network infrastructure, high-performance networks or may be completely heterogeneous in architecture.

This paper will begin with a short and general description of sending sequential jobs into the EGEE infrastructure. The approach for MPI jobs will then be described and compared to the sequential job approach. Some of the problems connected with parallel jobs will be listed. In the next two sections, the approach for sending MPI jobs in EGEE and I2G will be presented and compared to each other. Interesting improvements and features of I2G like the support for MPI tools or inter-cluster MPI will be presented as well. In the conclusion we summarize the results and give a short outlook for possible development of the MPI support.

2 SEQUENTIAL JOBS IN GRID INFRASTRUCTURES

In most Grid infrastructures, single processor jobs are still dominating the computer usage, although with the advance of multi-core architectures, increase in computing performance will depend on the parallelization of existing applications.

We give a very simplified description of the submission of sequential jobs in a grid. In order for a user to send a sequential program for execution on a Grid infrastructure, first of all, the user has to be a member of a Virtual Organisation (VO) in a grid. This means the user needs a valid certificate for this VO, or needs the VO authority to accept certificates issued by other authorities. Another requirement for the user is to have access to a so-called User Interface node of the grid. Job management is done from this node. A user then sends a job by describing in Job Description Language (JDL) a file to the Resource Broker (RB). The RB then matches the user's requirements against the available sites and sends them on a matched site. On this site, the Computing Element (CE) on his side sends the binary to a Worker Node (WN), where the execution of the sequential program takes place. The following example JDL file describes the job to execute the application `hostname` for EGEE:

```
Type          = "Job";
JobType       = "Normal";
Executable    = "/bin/hostname";
StdOutput     = "hostname.out";
StdError      = "hostname.err";
OutputSandbox = {"hostname.err","hostname.out"};
Arguments     = "-f";
RetryCount    = 7;
```

3 PARALLEL JOBS IN GRID INFRASTRUCTURES

Larger Grids consist of several clusters of SMPs, each consisting of dozens to hundreds of compute nodes, each with several sockets, each of which features up to four computing cores. Apart from other novel parallel programming paradigms, the Message Passing Interface (MPI) [10, 11] is the standard parallelization strategy for distributed memory machines, such as clusters of SMPs.

Our purpose now is to run an MPI application on the Grid. As MPI does not in itself impose any architecture requirements, an MPI job in principal can be executed on any architecture. In the JDL file, the user would specify the number of MPI processes to be launched for the application. Normally, the user would not oversubscribe the computer node, so that each MPI process would have its own single-core processor or its own core in a multi-core processor.

To make best usage of the given resources, a mechanism is required to reserve the resources and to convey the information described in the JDL file, as well as the input files required by the application. Within a single cluster, this resource allocation and fair distribution is handled by the job scheduler. Of course, Grid infrastructures must be aware of already existing scheduling mechanism on the back-end clusters, as described below.

3.1 Job Scheduler

Larger clusters with hundreds to thousands of compute nodes need some kind of scheduling to allocate compute nodes to users in a fair way. Users describe the applications requirements to the batch system in a description of the job similar to JDL. Allocation of those applications among the available computational resources is done by the so-called scheduler. Along comes the requirement to manage the execution once the application is started and once it has finalized or in case of an error. The batch system then handles the preparation of computational nodes, e.g. setting up networks, or mounting home directories prior to startup. Popular batch systems and job schedulers are Torque, an open source implementation of the Portable Batch System (PBS), the Sun Grid Engine (SGE) and the Load-Sharing Facility (LSF), etc. The MAUI scheduler is also often used in combination with the latter ones.

3.2 File Distribution

A specific problem to parallel applications is how to distribute the binaries as well as the input files to different worker nodes. Collecting the output files is also a similar problem. In general, there are multiple approaches.

The most popular approach within clusters is to use a networking file system, which offers access to files from multiple nodes. A common, yet not very scalable solution is to use the Network File System (NFS) protocol. However, from within multiple clusters within a Grid, NFS is not viable. Therefore Grid-specific solutions are discussed.

In the second approach the resource broker distributes the files specified in the input sandbox of the JDL to the first worker node of the nodes the job is allocated to. In the case that the different worker nodes are not using a shared filesystem, the other WNs will have to receive the files from the first WN, and in case the different WNs write separate output files, these files also have to be copied back to the first node.

Another approach is to use Storage Elements (SE) for input/output files. The transfer between SEs and WNs then has to be performed additionally.

3.3 Running the MPI Application

After the scheduler has allocated the job to some resources according to the user requirements, the execution of the MPI-parallel application takes place. The MPI Forum gives some recommendations on the mechanism of running MPI applications in the MPI-2 specification [11]. It recommends that *mpirun* is a portable and standardized script, while *mpiexec* is implementation-specific. The MPI forum recommends the availability of vendor-specific options only in *mpiexec*.

However, the different MPI vendors already were using *mpirun* in a non-portable and non-standardized way. Some of them started using *mpiexec* specifically for their MPI-2 implementation. In other implementations the two commands are identical. Even if a user assumes that he knows which command is needed, one still could not rely on specific options to *mpirun*/*mpiexec* – e.g. some MPI implementations support a specific scheduler or a mechanism for copying files to slave nodes, while others don't.

3.4 Making Things Work Together

The basic idea for a user to solve the problems that we mentioned above would be for the user to specify all information in its own script. By editing the script right before execution, the user is flexible about the execution of her job. In a next step, the script would be as generic as possible, so that the user would need no knowledge of the file system or scheduler installed on a site, letting the script extract this information. A further step is to install all scripts directly on the sites where the jobs are run. As we will see in the next sections, that is the path that EGEE and I2G follow. The latter project also adds important support into the resource broker.

4 MPI-SUPPORT IN I2G AND EGEE

In order to alleviate the problems of heterogenous cluster installations with regard to cluster network interconnection, the cluster's batch-job requirements and finally to the application's MPI-implementation requirements, several software components are integrated within the I2G-project to work within the Grid-environment.

MPI-Start is used as startup mechanism to adapt to the cluster installation and abstract the startup of applications with several MPI-implementations as backend, as described in Section 4.2.1.

Furthermore, with the Open MPI-project [2], HLRS is contributing to a high-performance MPI-implementation which still allows adaption to different infrastructures by supporting a variety of network interconnects and job schedulers for startup. All of these are easily loadable upon startup as plugins.

PACX-MPI (Section 4.2.5) – a solution for running an MPI application across different networks – was also implemented in HLRS. For Grids, it allows to utilize the usage of a very large number of CPUs on different clusters. Locally, the clusters use their performant MPI implementation, but they are connected through TCP connections over the Internet.

Marmot [9, 7] is an MPI correctness checker which was implemented at HLRS. It is based on the MPI Profiling Interface. Through its debug server, it is able to log runtime errors and warnings – e.g. about wrong data type usage, possible deadlocks, etc.

Different MPI applications from areas like high energy physics, nuclear fusion, astrophysics, medical applications and environment have been used. [4] describes one of the applications using MPI for environmental simulations. More information on all the applications is given in [8].

4.1 MPI in EGEE

In a previous approach to support MPI in EGEE one could directly specify a binary compiled with the MPICH version that was installed on several sites and specify "MPICH" as JobType. The broker then would issue a hard-coded script for the actual MPI job execution. This approach has multiple disadvantages: First, one is bound to one MPI implementation. Second, there is no flexibility at all: every minor change to the script from the broker is only applied after months of testing and validation. Clearly, a more flexible approach was needed, and MPI-Start was accepted as a solution.

Currently, in EGEE you can use a JDL like the following:

```
JobType       = "MPICH";
NodeNumber    = 16;
Executable    = "mpi-start-wrapper.sh";
Arguments     = "mpi-test OPENMPI";
StdOutput     = "mpi-test.out";
StdError      = "mpi-test.err";
InputSandbox  = {"mpi-start-wrapper.sh","mpi-hooks.sh","mpi-test.c"};
OutputSandbox = {"mpi-test.err","mpi-test.out"};
Requirements  =
 Member("MPI-START", other.GlueHostApplicationSoftwareRunTimeEnvironment)
&& Member("OPENMPI", other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

The mpi-start-wrapper.sh script sets some variables for MPI-Start (introduced in section 4.2.1). Also, the MPI implementation required in this case is Open MPI. The specified job type "MPICH" is misleading – it does not require the usage of MPICH, but at the moment it has to be used for parallel jobs. In some problem cases, the scheduler also has to be specified additionally. For more information refer to [12].

The main problem here is that there is no resource broker support other than the reservation of multiple nodes. The user manually sets all of the MPI-Start environment variables.

For a better understanding, Figure 1 (left) shows how the parts of the parallel job are specified and interpreted:

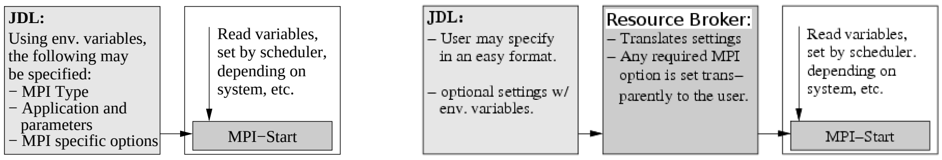
JDL:
Using env. variables, the following may be specified:
– MPI Type
– Application and parameters
– MPI specific options
Read variables, set by scheduler, depending on system, etc.
MPI–Start
JDL:
– User may specify in an easy format.
– optional settings w/ env. variables.
Resource Broker:
– Translates settings
– Any required MPI option is set trans– parently to the user.
Read variables, set by scheduler. depending on system, etc.
MPI–Start

Fig. 1. MPI-Start on EGEE (left) and I2G (right)

4.2 Int.EU.Grid

4.2.1 MPI-Start

In the previous sections we discussed the problems that can arise with MPI implementations, schedulers or file distribution. Other more application specific stuff like precompilation or collection of post-run results is also relevant. To resolve all of these problems, within the I2G project [5] a set of scripts were developed: MPI-Start. The main goal of MPI-Start is to support running MPI applications on such a challenging infrastructure as the Grid. The scripts are written in a modular way: there is a core and around it there are different plugins: MPI implementations, schedulers and file distribution method each have a set of plugins, as may be seen in Figure 2. Additionally, MPI-Start enables other features like user-defined scripts(hooks) to be run before or after the execution of the binary. Also, a very detailed debugging information is integrated for easier error detection. According to the detected plugins, the correct command is generated and executed to run the MPI program.

MPI-Start has two types of settings: some are the settings specified by the I2G cross broker [3] or the user directly through environment variables (e.g. MPI implementation to be used, hook methods to be called etc.), and others are through some kind of detection mechanism (file system for file distribution, scheduler, etc). The resulting command is executed.

In fact, it is difficult to categorize the tasks MPI-Start is limited to: everything that enables and improves running MPI applications is acceptable. The latest tendency in the development of MPI-Start is to allow the user comfortable usage of MPI tools for profiling or debugging. For some of those tools, it is necessary to have direct control of the runtime command (e.g. tracing tools) or some preconfiguring is necessary (like setting dynamic library paths). Support for one MPI correctness checker (Marmot) and one MPI tracing tool (mpitrace) has been added to I2G and successfully tested.

One challenge about the integration of tool support into MPI-Start is the loss of flexibility in some cases. In this respect, tools that are not highly customizable or are configured through environment variables are easily configured with MPI-Start support. Other classical Unix command line tools giving much flexibilty to the user should probably be used as part of the hooks mechanism and not directly integrated into MPI-Start.

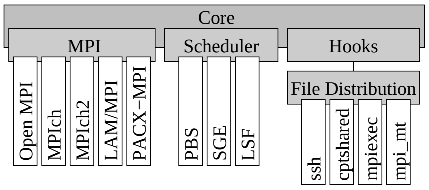

Fig. 2. MPI-Start architecture

4.2.2 MPI-Trace Support in I2G

The MPI tool usage on the I2G infrastructure can be exemplary seen on the example of mpitrace. Mpitrace [1] is a library for generating trace files for MPI applications for the Paraver analyzer. Although it would be possible to add this support on the user part as a set of pre- and post-run hooks, this solution could be much improved. Currently, MPI-Start offers support for trace file generation by two steps: first, you should compile your binary with mpitrace support; then you have to specify it is to be used by an extra flag to the JDL file. Also, you have to specify an output file which has a standardized name depending on the executable name. MPI-Start then automatically sets the necessary environment variables for mpitrace and after the execution, the necessary collection of the files on every machine and the generation of a global trace file are performed.

Of course, the resulting trace file has to be viewed and analyzed with Paraver. A user can do this on a local machine with an installation of the analyzer.

In the context of the Grid, such tracing information could be particularly interesting: an MPI application could depend not only on the algorithm but possibly also on the cluster the job is run on. Thus, comparing information about different clusters could help the user understand which performance problems are possibly algorithm-specific and which are only specific to a site configuration.

4.2.3 Marmot in I2G

Like we mentioned earlier, Marmot delivers relevant information for MPI application runs that can hint for programming errors at the MPI communication level. The motivation specifically for the Grid is to allow the expertise of Marmot to be used also by the application users of the infrastructure (usually engineers) to detect problems without being MPI experts. Marmot was used in a few runs of the Fusion application to verify it works without errors. For other applications, Marmot is available as well.

4.2.4 RB support for parallel jobs

In I2G, an essential step for supporting parallelism is the extension of the broker. On the user level, this can be first noticed by the syntax of the JDL file and the missing wrapper script for MPI-Start – it is not needed. The job type here is "Parallel" and the MPI implementation and executable (if the compilation is not delayed until before execution) can be directly specified.

```
Executable          = "mpi-test";
JobType             = "Parallel";
SubJobType          = "openmpi";
NodeNumber          = 16;
StdOutput           = "mpi-test.out";
StdError            = "mpi-test.err";
InputSandbox        = "mpi-test";
OutputSandbox       = {"mpi-test.err","mpi-test.out"};
```

However, the added value is mainly in the new functionality for parallel jobs that would not be available without the extension of the broker. The next section will introduce an inter-cluster MPI approach which is only possible through the cooperation of these components.

4.2.5 PACX-MPI

PACX-MPI [6] was developed at HLRS in order to allow one MPI application to be run on different MPI implementations on different hardware architectures. The software adds two additional MPI processes (daemons) for every MPP/cluster. The communication between the clusters is done over TCP connections and format conversions are done internally by the library. Using PACX-MPI simply requires recompiling the MPI code with the corresponding wrappers. The application must not even know it is being executed in a distributed heterogeneous environment over several clusters of the Int.EU.Grid.

An important part of using PACX-MPI is the definition of the metacomputer to run the MPI application on. In case we want to run the MPI application on two clusters which have no knowledge of each other before the application is run, a mediator has to be defined – the clusters contact it and get the connection information. This scenario is also used for running inter-cluster MPI applications in the

Int.EU.Grid. We will summarize how such an MPI application is enabled through all of the infrastructure components.

After that, the user specifies "pacx-mpi" as the SubJobType. Currently, the underlying MPI implementation is Open MPI on all sites. The user also specifies how many MPI processes are needed. Note that these processes are the application processes the user is interested in. After a modified matchmaking process takes place, possibly two sites supporting PACX are chosen.

Here, the elegance of the approach shown in Figure 1 (right) becomes obvious: the resource broker is able to specify environment variables for MPI-Start. These variables contain the information – where the startup server (the mediator mentioned above) is run, where the additional MPI processes for the inter-cluster communication should be run on the specific site, and also how the site is identified among all sites running MPI jobs. MPI-Start then acts correspondingly – it adds the extra daemons on the specified node, and it also specifies the mediator in its metacomputer configuration file. The whole process remains transparent to the user.

5 CONCLUSION

This paper presented how the MPI support for parallel applications has been improved on the Grid and in I2G in particular. The better support was in part delivered by the MPI-Start suite, but also by the resource broker and the way it communicates to MPI-Start. An example of how this support improves the grid was shown for the case of inter-cluster MPI – PACX-MPI. Recent extensions of MPI-Start also allow the use of MPI tools for debugging or profiling. This opens up the possibility for not simply running MPI applications, but also for a better understanding of performance and correctness issues when running those applications on the Grid. This includes a deeper understanding of how different sites with different characteristics behave for parallel applications.

REFERENCES

[1] Badia, R. M.—Escalé, F.—Gabriel, E.—Gimenez, J.—Keller, R.—Labarta, J.—Mueller, M. S.: Performance Prediction in a Grid Environment. Lecture Notes in Computer Science (LNCS), Vol. 2970, pp. 257–264, February 2003.

[2] Gabriel, E. et al.: Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In D. Kranzlmueller, P. Kacsuk, and J. J. Dongarra, editors, Proceedings of the 11th European PVM/MPI Users' Group Meeting, Vol. 3241 of Lecture Notes in Computer Science (LNCS), pp. 97–104, Budapest, Hungary, September 2004, Springer.

[3] Heymann, E.—Senar, M. A.—Fernández, E.—Fernández, A.—Salt, J.: Managing MPI Applications in Grid Environments. In 2nd European Across Grids Conference (AxGrids), Vol. 3165 of Lecture Notes in Computer Science, pp. 42–50, Springer, 2004.

[4] Hluchy, L.—Habala, O.—Tran, V.—Gatial, E.—Maliska, M.—Simo, B.—Slizik, P.: Collaborative Environment for Grid-based Flood Prediction. In Computing and Informatics, Vol. 24, 2005, No. 1, pp. 87–108.

[5] Int.EU.Grid Homepage. `http://www.interactive-grid.eu/`.

[6] Keller, R.—Liebing, M.: Using PACX-MPI in MetaComputing Applications. In Proceedings of the 18th Symposium Simulationstechnique, Erlangen, Germany, September 2005.

[7] Krammer, B.—Resch, M. M.: Correctness Checking of MPI One-Sided Communication Using Marmot. In B. Mohr, J. Larsson Trff, J. Worringen, and J. J. Dongarra, editors, Proceedings of the 13th European PVM/MPI Users' Group Meeting, Vol. 4192 of Lecture Notes in Computer Science (LNCS), pp. 105–114, Springer, September 2006.

[8] Marco, J.: The Interactive European Grid: Project Objectives and Achievements. See this volume.

[9] Marmot Homepage. `http://www.hlrs.de/organization/amt/projects/marmot/`.

[10] Message Passing Interface Forum. MPI: A Message Passing Interface Standard, June 1995, `http://www.mpi-forum.org`.

[11] Message Passing Interface Forum. MPI-2: Extensions to the Message-Passing Interface, July 1997, `http://www.mpi-forum.org`.

[12] MPI on EGEE. `http://egee-docs.web.cern.ch/egee-docs/uig/development/uc-mpi-jobs_2.html`.

Rainer Keller is working at HLRS in several projects, such as Open MPI and PACX-MPI. These software packages are being developped in several european and national projects, such as int.eu.grid. He is heading the working group application, models and tools.