Parallel Job Support in the Spanish NGI

Enol Fernández del Castillo

Instituto de Física de Cantabria, CSIC-UC, Spain
enolfc@ifca.unican.es

Abstract. Execution of parallel applications on a Grid environment is a challenging problem that requires the cooperation of several middleware tools and services. Although current grid middleware stacks have a basic level of support for such applications, this support is very limited. Users face a very heterogeneous environment lacking any standards for managing their jobs. We present a complete framework for the execution of parallel applications in grid environments that hides the underlying complexity with a simple interface to run the applications. The framework is composed of a unified interface layer for starting parallel applications and a grid scheduler that provides transparent and reliable support for such types of applications. The framework is focused on MPI applications, but extensible to other types of parallel applications.
Keywords: Grid, MPI, parallel jobs.

1 Introduction

Execution of parallel applications in clusters and HPC systems is commonplace. The Message Passing Interface (MPI) [18] programming paradigm is a widely used standard for parallel application in scientific and industrial domains, such as meteorology, neuroscience, industrial engineering, etc. MPI support at individual resource centers, ranging from large HPC centers to smaller cluster sites, is at a mature level, and is consequently well understood. Typically, multiple MPI work environments may be installed to provide advanced features accommodating the local users needs.

Grid infrastructures are mainly used for the execution of collections of sequential jobs [11] although most infrastructures are composed of clusters where execution of parallel applications is possible. Executing such applications is a challenging problem that requires the cooperation of several middleware tools and services. Most grid middleware implementations have some support for such applications, but this support is usually limited to the possibility of allocating a set of nodes. Starting an MPI application requires dealing with low level details of a set of elements:

– Local Resource Management System (LRMS). Clusters are normally managed by a scheduling system that allocates the compute nodes to users in a fair way. Popular LRMS are SGE [8], PBS [2], LSF [22] and Condor [14]. Each system has particular ways to manage and interact with the nodes of the cluster.

– File distribution. The execution of a parallel application requires the distribution of binaries and input files into the different nodes involved in the execution. Collecting the output is a similar problem. There are multiple approaches, like using network file-systems or copying the files to each node.
– Application execution. Each parallel library or framework has different ways of starting the application. Moreover, for a given framework there may be differences depending on the LRMS or file distribution method used in the execution environment. In the case of MPI, the MPI Forum [17] gives recommendations on the mechanism for running the applications in the MPI-2 specification [18]: *mpirun* should be a portable and standardized script, while *mpiexec* is implementation specific. However, the different MPI vendors already were using *mpirun* in a non-portable and non-standardized way. In some implementations both *mpiexec* and *mpirun* are identical, while other implementations do not support one of them.

Users face this complexity, aggravated by the heterogeneity of grid environments, with little or no support from middleware. In this paper we present a framework that shows to the user a unique and simple interface for the execution of parallel applications. The framework is composed by two elements: MPI-Start – a unified layer for starting the applications – and the CrossBroker [6] – a Grid Resource Management System (GRMS) that provides scheduling services for parallel and interactive applications. The services were originally developed as part of the CrossGrid [3] and int.eu.grid [15] projects and now are actively maintained and further developed in the Spanish NGI.

In Section 2, a description of the unified layer for parallel applications is given. Section 3 gives an overview of the CrossBroker and presents the newer features for executing jobs. In Section 4, we give some conclusions and an outlook of future work.

2 Unified layer for starting parallel applications

In order to provide a uniform interface for running MPI applications in grid environments, MPI-Start project was started in the frame of int.eu.grid [15]. MPI-Start is a unique layer that hides the details of the resources and application frameworks (such as a MPI implementation) to the user and upper layers of the middleware as shown in Figure 1. The use of such layer removes any dependencies in the middleware related to starting parallel applications.

One of the main advantages of MPI-Start is the ability to detect and use site-specific configuration features without user intervention – such as the batch scheduler and the file system at the site. Users just need to specify their executable and arguments to run their application, while at the same time, advanced users can easily customize the behavior and have a tighter control on the application execution. Support of simple file distribution and compilers is also included in this tool.

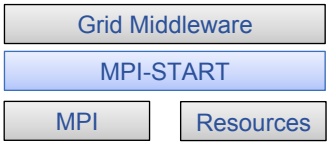

Fig. 1. Unified layer for parallel applications

2.1 MPI-Start Architecture

MPI-Start is a set of scripts that ease the execution of MPI programs by using a unique and stable interface to the middleware. The scripts are written in a modular way: there is a core and around it there are different plug-ins as shown in Figure 2. Three main frameworks are defined in the MPI-Start architecture:

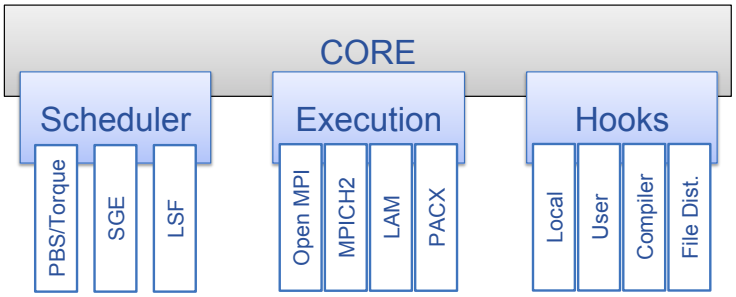

Fig. 2. MPI-Start architecture

- Scheduler framework. Every plug-in for this framework provides support for different Local Resource Management Systems (LRMS). They must detect the availability of a given scheduler and generate a list of machines that will be used for executing the application. The supported LRMSs are SGE, PBS and LSF.
- Execution framework. These plug-ins set the special parameters that are used to start the MPI implementation. It is not automatically detected, hence the user or CrossBroker must explicitly specify which MPI flavor will be used to execute the application. There are plug-ins implemented for Open MPI [7], MPICH [10] (including MPICH-G2 [12]), MPICH2 [9], LAM-MPI [20] and PACX-MPI [13].
- Hooks framework. In this framework any additional features that may be needed for the job execution is included. It is customizable by the end-user without the need of any special privileges. File distribution and compiler support is included in the hooks framework.

The hooks framework opens the possibility of customizing the behavior of MPI-Start. There are four classes of hooks: local-site, compiler, end-user, and file distribution. Local-site hooks allow the system administrator to set any configuration specific to the site that are not set automatically by MPI-Start. In order to obtain good performance and to assure that the binaries will fit the available resources, MPI jobs should be compiled with the local MPI implementation at each site. MPI-Start compiler support in the hooks framework detects the system architecture and sets the adequate compiler flags to build the binaries. Users can compile their application using the end-user hook, this is a shell script where the exact command line for building the application can be specified. Any input data preparation can be also done in that script.

One of the most important features in MPI-Start is the file distribution hook. Most MPI implementations need application binaries to exist in all the machines involved in the execution. This hook is executed once all the previous hooks have finished correctly and assures that all the needed files are available for the job at the different hosts. Prior to file distribution, a test detects the availability of a network file system. If this kind of file system is not found, MPI-Start will copy the files using the most appropriate method according to the site configuration. In the current implementation, the following methods are supported: OSC Mpiexec [21], ssh, mpi-mt (a MPI tool for copying files with MPI-IO) and copying the files to a shared area defined by the administrator.

Figure 3 shows the flow of execution of MPI-Start. The first step is the detection of the scheduler plug-in and the creation of a *machinefile* with the list of hosts in a default format. This is done automatically by checking the execution environment of the job.

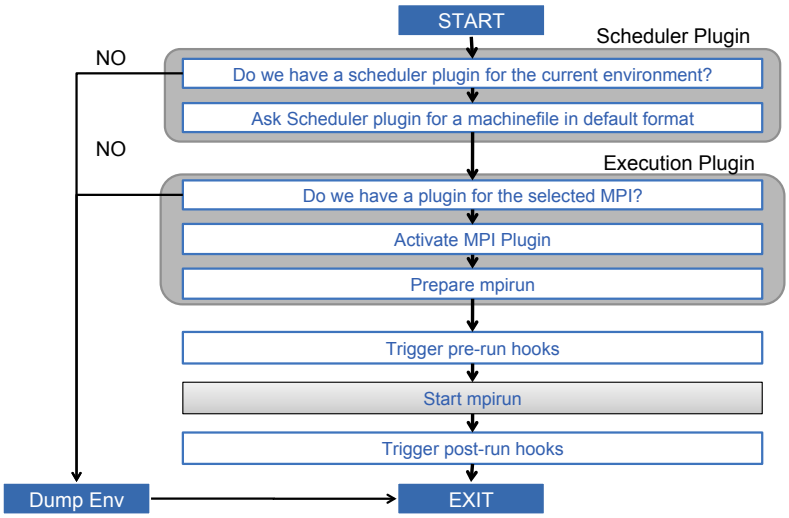

Fig. 3. MPI-Start execution flow

The next step is the selection of the execution plug-in according to the value specified by the user. The plug-in will activate the MPI implementation and prepare the correct command line to start the job. MPI-Start is able to detect any special tools available in the site, such as OSC Mpiexec, that may be used for a better startup of the application. If any of the two first steps fail, the program will exit and dump the execution environment to aid in debugging the problem.

Once the environment is ready to execute the MPI type selected, the hooks framework is executed in the following order: compiler, local-site, end-user and file distribution. If any of the hooks is not found it will be skipped. The following step is the actual execution of the application using the appropriate command line created by the execution plug-in and potentially modified by the hooks. The last step is the post-run hooks that allow fetching results or output processing. The post-run hooks executed are local-site, end-user and file distribution.

MPI-Start is implemented in a way that is easy to use even if it is not installed with special privileges. It can be included with the input files of the job and be executed from any location of the file system. The use of shell scripts for its implementation gives to MPI-Start good portability across systems. It also includes exhaustive debugging features. With different levels of debugging verbosity, MPI-Start can give from a totally silent output to a complete trace of all the steps performed during its execution.

The interface between the user and MPI-Start is a set of environment variables listed in Table 1. The simplest job just needs to specify the `I2G_MPI_APPLICATION`, `I2G_MPI_APPLICATION_ARGS` and `I2G_MPI_TYPE`, with the location of the application, its arguments and the MPI implementation that will be used. The rest of variables allow further customization of the job execution. `I2G_MPI_PRE_RUN_HOOK` and `I2G_MPI_POST_RUN_HOOK` variables let the user specify the hooks. Interactive applications may use the `I2G_MPI_PRECOMMAND` to specify a program that creates a channel of communcation between the user and the application. PACX-MPI applications that are executed across several clusters specify with `I2G_MPI_FLAVOUR`, `I2G_MPI_JOB_NUMBER`, `I2G_MPI_STARTUP_INFO` and `I2G_MPI_RELAY`, the details for establishing the communication between the different resources.

2.2 Extending MPI-Start for other parallel jobs.

Although MPI-Start was designed to start MPI applications in grid resources, other parallel application paradigms and libraries may be supported with the current architecture. New libraries or applications can be supported by creating new modules in the *Execution framework* as shown by the following examples:

Kepler Workflows. We have created a MPI-Start plug-in for the execution of workflows within a cluster using Kepler [16]. Kepler is a workflow engine where a set of *actors* are executed following rules defined by the user. The engine includes several of these *actors* for the most common actions, but it can also be extended by user provided ones. Two new actors were created: a first one that executes simultaneously the same application in a set of nodes, and a second one that

Variable	Meaning
I2G_MPI_APPLICATION	The application binary to execute.
I2G_MPI_APPLICATION_ARGS	The command line parameters for the application.
I2G_MPI_TYPE	The name of the MPI implementation to use.
I2G_MPI_PRECOMMAND	Allows running the MPI application "inside" another command.
I2G_MPI_FLAVOUR	Specifies which "sub-mpi" to use. In the case of a PACX-MPI job, this variable specifies the local MPI implementation to use.
I2G_MPI_PRE_RUN_HOOK	This variable can be set to a script which that defines the pre_run_hook function executed before the application starts.
I2G_MPI_POST_RUN_HOOK	This variable can be set to a script which that defines the post_run_hook function executed after the application finishes.
I2G_MPI_JOB_NUMBER	If a MPI job runs across multiple clusters this variable specifies which sub-job should be started on this cluster. The values are 0, 1, 2,... In the case of an MPI job that runs only inside the local cluster this variable is always 0.
I2G_MPI_STARTUP_INFO	This variable provides additional information for the MPI program. In the case of PACX-MPI, this variable specifies the connection information to the startup server.
I2G_MPI_RELAY	This variable specifies an FQDN of a host that can be used as relay/proxy host. In the case of PACX-MPI on this host the additional 2 proxy processes will be started. It's required that this host has out-bound connectivity and is accessable via ssh.

Table 1. MPI-Start environment variables.

selects only one of the available nodes for executing an application, while the rest of nodes can be used by different instances of the same actor for other applications.

Both actors make use the new plug-in. This plug-in takes profit of the automatic detection of the available nodes and file transfers provided by MPI-Start. Moreover, they make use of the batch system detection to select the most suitable way of starting the applications remotely. Whenever possible, native tools of the batch system are used (e.g. `qrsh` in SGE) and ssh is used as a fall-back method when no other tool is available. Once the application has ended the transfer of files from the executing nodes is also handled by MPI-Start file distribution hooks. The plug-in does not include any Kepler dependency and thus can be used by any other tool application that needs to start parallel jobs without communication in a set of nodes.

Hybrid MPI/OpenMP applications. Parallel applications using the shared memory paradigm are becoming more popular with the advent of multi-core architectures. MPI-Start default behavior is to start a process for each of the slots allocated for an execution. However, this is not suitable for applications using a hybrid architecture where several threads access to a common shared memory area in each of the nodes. For these cases, the MPI-Start execution framework was extended to start only one process per node. Before the exection of the user process, the number of slots allocated is exported to the environment variable `I2G_MPI_NODE_SLOTS` at each of the nodes. The value of this variable should be used by the user process as the maximun number of threads to spawn. In order to ease the use of OpenMP applications, the `OMP_NUM_THREADS` variable is also set to the same value.

3 Scheduling for parallel applications

In large-scale grids, grid scheduler services are essential to free the users from the cumbersome work of job handling. The CrossBroker is a GRMS focused on the execution of parallel and interactive jobs. It provides support for co-allocation of resources across different administrative domains, opening the possibility of running parallel applications which can efficiently use many processors, taking advantage of the different resources composing a grid. A complete description of the CrossBroker features for the execution of parallel and interactive jobs can be found in [6].

The CrossBroker simplifies the submission of applications using some of the most common MPI implementations; both for execution within a single cluster with Open MPI, MPICH or MPICH2 and using several clusters with PACX-MPI and MPICH-G2. The JDL [19] is augmented with a new type of jobs – *Parallel* – and a new attribute, – *SubJobType* – that allows the user to specify the MPI implementation to use. The scheduler automatically selects only sites that support the MPI flavor and assures that there are enough slots to run the job at the resources. Parallel jobs are executed using MPI-Start, which is transparently configured on behalf of the users to run their applications. All the environment variables that

serve as interface to MPI-Start are set to the appropriate values. Figure 4 shows an example of a parallel job that uses 12 processes with Open MPI.

```
Executable              = "fusion_app";
Arguments               = "-n";
JobType                 = "Parallel";
SubJobType              = "openmpi";
NodeNumber              = 12;
InputSandBox            = {"fusion_app"};
```

Fig. 4. JDL job description

The execution of large bunch of jobs and parameter sweep applications is common in grid environments. We have included in the CrossBroker the support of such applications with two new kind of jobs: *collection* and *parametric*. Collections are sets of independent jobs sharing similar requirements that can be matched in clusters according to some significant attributes and then tracked with a single handle. Using collections reduces significantly the time for submission to the CrossBroker (only one submission for several jobs) and simplifies the management of the jobs to the users. Parametric jobs allow the execution of applications using the parameter sweep technique. Each of the individual jobs that compose a collection or a parametric job can be parallel jobs and use the interactivity features of the CrossBroker.

The implementation of those jobs is compatible with the JDL specification as implemented by the gLite WMS [1]. However, in order to give a more power to the end-user, we have extended the parametric job definition. The original JDL specification only permits the definition of a single parameter per job. Three different attributes determine the parameter values: *Parameters*, *ParameterStart* and *ParameterStep*. For each of these jobs, the `_PARAM_` keyword is substituted in the user description with the value of the parameter. The list of parameters can also be specified using only the *Parameters* attribute. In the case of numeric parameters, N different jobs would be generated, where

$$N = (Parameters - ParameterStart)/ParameterStep \tag{1}$$

In the CrossBroker, we have extended the specification to allow using more than a single parameter in each job. Each new parameter must have an identifier that is appended at the JDL attributes in the following form: *Parameters_id*, *ParameterStart_id*, *ParameterStep_id*, with *id* being the parameter identifier, similarly the keyword `_PARAM_id` is substituted for generating the individual jobs. The complete parameter space is explored, generating as many jobs as possible combinations of parameters.

Figure 5 shows an example of a parametric job with two different parameters. The first parameter is called `A` and takes the values *alpha* and *beta*, while the second one is called `B` and is defined by a range: 2 different values, starting from 0 and with

a step of 1. All the combinations of parameters are evaluated, resulting in a total of four different jobs. The jobs would have the following values for `StdInput`: *input-alpha-0.txt*, *input-beta-0.txt*, *input-alpha-1.txt* and *input-beta-1.txt*. Similar values would be generated for `StdOutput`, `StdError` and `InputSandBox` attributes.

```
JobType = "Parametric";
Executable = "myexec";
StdInput = "input-_PARAM_A_-_PARAM_B_.txt";
StdOutput = "output-_PARAM_A_-_PARAM_B_.txt";
StdError = "error-_PARAM_A_-_PARAM_B_.txt";
Parameters_A = {alpha, beta};
Parameters_B = 2;
ParameterStart_B = 0;
ParameterStep_B = 1;
InputSandbox = {"input-_PARAM_A_-_PARAM_B_.txt"}
```

Fig. 5. Parametric job with two parameters

4 Conclusions and Future Work

MPI-Start and the CrossBroker provide a complete framework for the execution of parallel applications on the Grid. The unique interface of MPI-Start simplifies the task of starting the jobs by handling transparently the low level details of the different Local Resource Management Systems, execution frameworks and file distribution methods. The most common batch systems and MPI implementations are supported by MPI-Start and its modular architecture allows the easy extension of the tool to support new kind of parallel jobs, such as Kepler Workflows and hybrid MPI/OpenMP applications. MPI-Start has become the official way of starting the jobs in EGEE, one of the biggest grid infrastructures worldwide. This software will be further developed in EMI [5], providers of middleware for the European Grid Initiative [4].

The CrossBroker is a Grid Resource Management System focused on parallel and interactive applications. Its tight integration with MPI-Start provides users with an easy way of running their applications on the infrastructure. We have extended the CrossBroker with new kind of jobs: collections and parametric. These are suited for the execution of embarrassingly parallel applications with no communication between the tasks and parameter sweep applications. The parametric jobs support is enhanced from the standard JDL specification and gives the user the possibility to submit complex jobs where more than one parameter is explored.

The availability of multi-core architectures opens the possibilities of new types of parallel jobs and will change the interaction of applications with the batch systems. Future work will investigate how this changes could affect MPI-Start and the integration of new types of applications in the tool. Currently the middleware is

only capable of allocating a given number of cores, but no details of the distribution of those cores can be specified. Better ways of requesting the adequate mapping for the user applications will be explored in the CrossBroker and rest of the middleware involved in the execution.

Acknowledgements

The authors acknoledge support of the European Comission FP7 program, under contract number 211804 through the project EUFORIA (http://www.euforia-project.eu).

References

1. P. Andreetto et al. "The gLite workload management system", *Journal of Physics: Conference Series*, **volume 119(6)**, (2007).
2. A. Bayucan, R. L. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten. "Portable batch system: External reference specification", Technical report, MRJ Technology Solutions, (1999).
3. CrossGrid Project http://www.eu-crossgrid.org/
4. European Grid Initiative (EGI) http://www.egi.eu/
5. European Middleware Initiative (EMI) https://twiki.cern.ch/twiki/bin/view/EMI
6. E. Fernández, A. Cencerrado, E. Heymann, and M. A. Senar. "CrossBroker: A Grid Metascheduler for Interactive and Parallel Jobs", *Computing and Informatics*, **volume 27**, pp. 187–197, (2008).
7. E. Gabriel et al. "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation". *Lecture Notes in Computer Science*, **volume 3241**, pp. 97–104, (2004).
8. W. Gentzsch. "Sun Grid Engine: towards creating a compute power grid", *Proceedings of the first IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 35–36, (2001).
9. W. Gropp. "MPICH2: A new start for MPI implementations", *Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pp. 7, (2002).
10. W. Gropp, E. Lusk, N. Doss, and A. Skjellum. "A high-performance, portable implementation of the MPI message passing interface standard", *Parallel Computing*, **volume 22(6)**, pp. 789–828, (1996).
11. A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H.J. Epema. "The Grid Workloads Archive", *Future Generation Computer Systems*, **volume 24(7)**, pp. 672–686, (2008).
12. N. T. Karonis et al. "MPICH-G2: A grid-enabled implementation of the message passing interface", *J. Parallel Distrib. Comput.*, **volume 63(5)**, pp. 551–563, (2003).
13. R. Keller, E. Gabriel, B. Krammer, M. S. Müller, and M. M. Resch. "Towards efficient execution of MPI applications on the grid: Porting and optimization issues", *Journal of Grid Computing*, **volume 1(2)**, pp. 133–149, (2003).
14. M. Litzkow, M. Livny, and M. Mutka. "Condor - a hunter of idle workstations", *Proceedings of the 8th International Conference of Distributed Computing Systems*, (1988).

15. J. Marco et al. "The Interactive European Grid: Project Objectives and Achievements" *Computing and Informatics*, **volume 27**, pp. 161–173, (2008).
16. T. McPhillips, S. Bowers, D. Zinn, and B. Ludäscher. "Scientific workflow design for mere mortals", *Future Generation Computer Systems*, **volume 25**, pp. 541–551, (2009).
17. MPI Forum. http://www.mpi-forum.org/.
18. Message Passing Interface Forum. "MPI: A Message Passing Interface standard", (1995).
19. F. Pacini and A. Maraschini. "Job Description Language (JDL) attributes specification", Technical Report 590869, EGEE Consortium, (2006).
20. J. M. Squyres. "A component architecture for LAM/MPI", *Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pp. 379–387, (2003).
21. P. Wyckoff. OSC Mpiexec. http://www.osc.edu/ djohnson/mpiexec.
22. S. Zhou. "Lsf: Load sharing in large-scale heterogeneous distributed systems", *Proccedings of the Workshop on Cluster Computing*, (2002).