Computing and Informatics, Vol. 31, 2012, 149–160

SUPPORT TO MPI APPLICATIONS ON THE GRID

Enol Fernández-del-Castillo

Instituto de Física de Cantabria
Edificio Juan Jordá
Avenida de los Castros s/n
39005 Santander, Spain
e-mail: enolfc@ifca.unican.es

Abstract. The current middleware stacks provide varying support for the Message Passing Interface (MPI) programming paradigm. Users face a complex and heterogeneous environment where too many low level details have to be specified to execute even the simplest parallel jobs. MPI-Start is a tool that provides an interoperable MPI execution framework across the different middleware implementations to abstract the user interfaces from the underlying middleware and to allow users to execute parallel applications in a uniform way, thus bridging the gap between HPC and HTC. In this work we present the latest developments in MPI-Start and how it can be integrated in the different middleware stacks available as part of EMI, providing a unified user experience for MPI jobs.

Keywords: Grid, MPI, parallel jobs

1 INTRODUCTION

Execution of parallel applications in grid environments requires the cooperation of several middleware tools and services. Two main phases can be identified in the submission of such applications: the allocation of nodes where the user job will run, and the execution of the application using those allocated nodes. The middleware support for MPI applications is usually limited to the possibility of allocating a set of nodes. The user still needs to deal with low level details related to the actual execution of the jobs that make the task non trivial. Furthermore, the heterogeneity of resources available in Grid infrastructures aggravates the complexity that users must face to run their applications.

The European Middleware Initiative (EMI) [8] project is the main developer of grid middleware in Europe by supporting the development and integration of three different middleware stacks for the the execution of jobs: ARC [5], gLite [17], and UNICORE [6]. All of them have some support for the execution of parallel applications. However, all of them have different approaches that prevent users from easily moving from one stack to another.

MPI-Start [4] is also being developed in the context of the EMI project. MPI-Start is a unique layer that hides the details of the resources and application frameworks to the user and upper layers of the middleware. By using a modular and pluggable architecture it manages the details of several elements for the user: from Local Resource Management System (LRMS) to the specific syntax to start an application for a given MPI implementation.

In this paper we describe the current support for MPI jobs in the different EMI middleware stacks and how MPI-Start may be integrated with all of them in order to provide a unified user experience across the different stacks. In Section 2, a description of the middleware support for allocation of nodes is given. Section 3 describes MPI-Start and the integration with the middleware stacks. In Section 4, we describe the monitoring probes for MPI in the EGI [7] Infrastructure. Finally, in Section 5 we give some conclusions and an outlook of future work.

2 JOB SUBMISSION

Prior to the execution itself, the parallel application must be submitted to a grid middleware that will create a work item on a Local Resource Management System (LRMS). This middleware is usually referred to as Computing Element (CE). EMI provides implementations of such CE in three different middleware stacks: ARC, gLite and UNICORE. All of them provide some support for parallel applications although the level of control for the job definition varies from one stack to other.

ARC provides the ARC-CE, which uses the *Extended Resource Specification Language* (xRSL) language [9] for defining the jobs. In order to submit a parallel application, the `count` attribute is used to specify the number of slots that must be allocated for the application. ARC provides also the *Runtime Environments* (RTE), that allow the site administrator to define an environment for the execution of specific applications. The usual way of supporting an MPI implementation in ARC is by defining a RTE for a specific MPI implementation. The user must then write a script that uses a set of predefined variables to start the application. Listing 1 shows an example of a 16 processes MPI application that is submitted to an ARC-CE using the `OPENMPI-1.3` Runtime Environment, the script that the user should provide for its execution is shown in listing 2. Note that the user builds the command line required to start the job, therefore the user must know the specific syntax for the MPI implementation used.

```
&(executable="runopenmpi.sh")
(executables=("hello-ompi.exe" "runopenmpi.sh"))
```

```
(count="16")
(inputfiles=("hello-ompi.exe" "runopenmpi.sh"))
(stdout="std.out")
(stderr="std.err")
(runtimeenvironment="ENV/MPI/OPENMPI-1.3/GCC64")
```

Listing 1. ARC parallel job description

```
#!/bin/sh

$MPIRUN -np $NSLOTS ./hello-ompi.exe
```

Listing 2. ARC parallel job script

gLite provides the CREAM [1] as Computing Element. The language used to describe jobs in CREAM is the *Job Description Language* (JDL) [19]. The user has several ways of defining a parallel job. The most basic case is using the `CPUNumber` attribute, that determines the total number of slots to be allocated by the CE. Advanced placement of the processes on the physical hosts can be also requested with the following attributes:

SMPGranularity This value determines the number of cores any host involved in the allocation has to dedicate to the application.

WholeNode Indicates whether whole nodes should be used exclusively or not.

NodeNumber This integer value indicates the number of nodes the user wishes to obtain.

The CREAM does not provide any additional support for the job execution. However, MPI-Start is usually available in gLite sites to start parallel jobs. Details on MPI-Start are given in the next Section. Listing 3 shows an example of a MPI application with 16 processes submitted to a CREAM.

```
JobType = "Normal";
CPUNumber = 16;
Executable = "starter.sh"
InputSandbox = {"starter.sh", "hello-ompi.exe"};
StdOutput = "std.out";
StdError = "std.err";
OutputSandbox = {"std.out", "std.err"};
```

Listing 3. gLite parallel job description

The `starter.sh` script invokes MPI-Start after setting some variables that determine which application the user wants to start. An example script is shown in Listing 4.

```
#!/bin/bash

export I2G_MPI_APPLICATION=hello-ompi.exe
```

```
export I2G_MPI_TYPE=openmpi

$I2G_MPI_START
```

Listing 4. gLite parallel job script

In the case of UNICORE jobs, users can describe their jobs using a JSON representation of the *Job Submission Description Language* (JSDL) [2] format. Parallel jobs are described using the `Resources` attribute. This is a complex attribute that may contain the number of requested slots with the `CPUs` attribute, or advanced placement of the processes with the `CPUsPerNode` attribute that indicates the number of cores in the host involved in the allocation, and the `Nodes` attribute that indicates the total number of nodes the user wishes to use. The execution of the applications is handled with the *Execution Environments*. Site administrator defines as many Execution Environments as needed with the specific details of each kind of job the user may execute at the site. A template based language is used to describe these Execution Environments. Listing 5 depicts the description of an MPI job with 16 processes submitted to UNICORE. Note that there is no need to specify any user script, the Execution Environment takes care of all the details.

```
{
  Executable: "./hello-ompi.exe",
  Imports: [
    {From: "/myfiles/hello.mpi", To: "hello-ompi.exe"},
  ],
  Resources: {
    CPUs: 16,
  },
  Execution environment: {
    Name: OpenMPI,
    Arguments:
      Processes: 16,
    },
  },
}
```

Listing 5. UNICORE parallel job description

3 MPI-START

As shown in the previous section, users are able to submit and execute parallel jobs using the current middleware stacks. However, each of them provides different level of support and abstraction that turns the migration from one implementation to another hard for most users. MPI-Start provides an abstraction layer that simplifies the execution of the jobs in heterogeneous systems available in grid environments. MPI-Start takes care of the following details for the user in an automatic way:

- Local Resource Management System (LRMS). Each system has particular ways to manage and interact with the nodes of the cluster. MPI-Start automatically detects and prepares the list of machines for SGE [11], PBS/Torque [3], LSF [22], Condor [18] and Slurm [21] batch systems.
- 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. MPI-Start has a file distribution hook that detects if a shared filesystem is available. In the case of not being available, MPI-Start distributes binaries and input files into the execution hosts using the most appropriate method.
- Application compilation. In order to obtain good performance and to assure that the binaries will fit the available resources, MPI jobs may need to be compiled with the local MPI implementation at each site. MPI-Start checks the compilation flags in the system and assures that users are able to compile their applications.
- 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 different vendors use mpirun and mpiexec in a non-portable and non-standardized way. MPI-Start builds the command line for common available MPI implementations such as Open MPI [10], MPICH [12] (including MPICH-G2 [15]), MPICH2 [13], LAM-MPI [20] and PACX-MPI [16].

The latest developments of MPI-Start have introduced a new architecture for extensions, the ability to define the way the user logical processes are mapped in the physical resources and a complete review of the LRMS and Application Execution support. These developments will be available as part of the EMI-1 release, due in May 2011.

3.1 Hybrid MPI/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. In order to support more use cases, the latest report of MPI-Start includes support for better control of how the processes are started, allowing the following behaviors:

- Define the total number of processes to be started, independently of the number of allocated slots.
- Start a single process per host. This is the usual use case for hybrid jobs with MPI applications. MPI-Start prepares the environment to start as many threads as slots available in the host.

- Start a single process per CPU socket. In this case, a hybrid application would start as many threads as cores are available for each CPU.
- Start a process per CPU core, independently of the number of allocated slots.
- Define the number of processes to be started in each host, independently of the number of allocated slots at each host.

Figure 1 shows the different possible mappings for two hosts with two quad-core CPUs. In the per core case, there would be 16 MPI processes, numbered from $p0$ to $p15$, each assigned to one CPU core. In the case of using a per socket mapping, each host would have two different proceeses – $p0$ and $p1$ in the first host, $p2$ and $p3$ in the second host – and for each of these processes, 4 different threads ($t0$ to $t3$). Finally, if the per node mapping is used, only one processes would be started at each host – $p0$ in first host, $p1$ in the second one – and 8 threads would be started for each of them, numbered from $t0$ to $t7$.

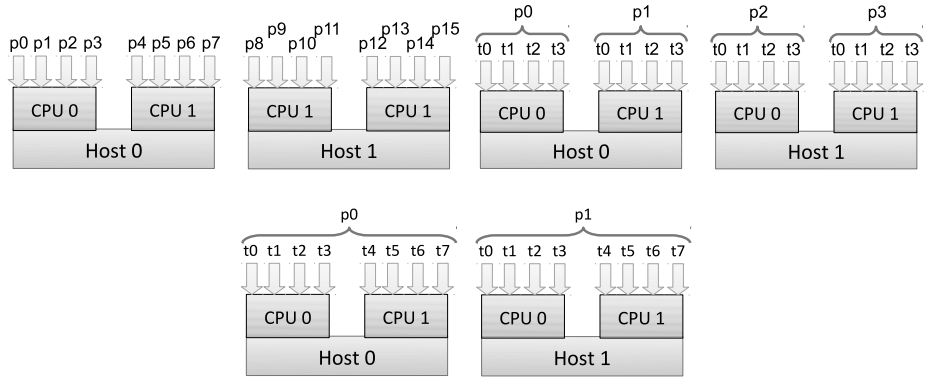

Fig. 1. MPI-Start mapping of processes: top left: per core; top right: per socket; bottom: per node.

Additionally, two new extensions (hooks in the MPI-Start terminology) have introduced support for OpenMP and CPU affinity. The OpenMP support defines in the environment the most appropriate number of threads to use by the applications given the machine configuration and the mapping selected. The affinity hook enables, if available, the processor and memory affinity features of the MPI implementation by creating mappings similar to the ones shown in Figure 1.

3.2 Integration with ARC and UNICORE

MPI-Start was originally developed for the integration with the gLite middleware, although the design and architecture of MPI-Start is completely independent of this middleware. For the EMI-1 release, we have integrated the tool with the ARC and

UNICORE middlewares by providing new Runtime Environments and Execution Environments that can be easily configured by the site administrators.

In the case of ARC, the definition of a Runtime Environment consists in the creation of a shell script that is invoked three times for any given execution: before the job is submitted, before the execution of the job itself and after the job has finished. Listing 6 shows the code for this RTE. Site administrators only need to define any special configurations that may have their site for MPI-Start. In the given example, by defining the variable `MPI_START_SHARED_HOME` to `yes`, the site admin is indicating to MPI-Start that it should not try to detect which kind of filesystem is available and assume a shared file system will be used. Users of this site would only need to require in their job description the `mpi-start` Runtime Environment and use MPI-Start as in a gLite site.

```
#!/bin/bash
parallel_env_name="mpi-start"
case "$1" in
0 ) # local LRMS specific settings, no action
  ;;
1 ) # user environment setup
  export I2G_MPI_START=/usr/bin/mpi-start
  export MPI_START_SHARED_HOME=yes
  ;;
2 ) # no post action needed
  ;;
* ) # everything else is an error
  return 1
  ;;
esac
```

Listing 6. ARC Runtime Environment

The definition of UNICORE Execution Environments is done using an XML file where the options and their rendering are described. In order to use MPI-Start in such way, we introduced the possibility of setting the parameters via command line arguments instead of environment variables. Listing 7 shows partially one example definition of an Execution Environment for MPI-Start. In the example the user can define the MPI implementation to use, the total number of processes, additional MPI-Start variables and enable the verbose output. A complete Execution Environment would provide additional options for controlling all the MPI-Start features.

```
<ExecutionEnvironment>
  <Name>mpi-start</Name>
  <Description>Run parallel applications</Description>
  <ExecutableName>/usr/bin/mpi-start</ExecutableName>
  <Argument>
      <Name>mpi type</Name>
```

```
      <IncarnatedValue>-t </IncarnatedValue>
      <ArgumentMetadata>
          <Description>MPI implementation</Description>
          <Type>string</Type>
      </ArgumentMetadata>
  </Argument>
  <Argument>
      <Name>Number of Processes</Name>
      <IncarnatedValue>-np </IncarnatedValue>
      <ArgumentMetadata>
          <Description>The number of processes</Description>
          <Type>int</Type>
      </ArgumentMetadata>
  </Argument>
  <Argument>
      <Name>MPI-Start Variable</Name>
      <IncarnatedValue>-d </IncarnatedValue>
      <ArgumentMetadata>
          <Description>
            Define a MPI-Start variable
            (e.g., "I2G_MPI_START_VERBOSE=1")
          </Description>
          <Type>string</Type>
      </ArgumentMetadata>
  </Argument>
  <Argument>
      <Name>Verbose</Name>
      <IncarnatedValue>-v</IncarnatedValue>
      <OptionMetadata>
          <Description>Be verbose</Description>
      </OptionMetadata>
  </Argument>
</ExecutionEnvironment>
```

Listing 7. UNICORE Execution Environment

With the integration of MPI-Start into the ARC and UNICORE approaches for job execution, users are provided with a unified user experience. They only need to specify the correct parameters to MPI-Start and can easily move from one middleware to other, or from one MPI implementation to other without worrying about the details of each of them. For example, a hybrid application that uses Open MP and Open MPI for execution, that needs to be compiled at the site using the MPI-Start hooks mechanism could be defined for the three middlewares as shown in Listings 8, 9 and 10. In the example the user defines the variable `MPI_USE_OMP` to activate the OpenMP support, it requires the execution of only one MPI process

per host with the `pnode` option and includes the hook `myhook.sh` for compilation before the actual execution. Note that the only differences are due to the description language of each middleware.

```
(Arguments="-t openmpi -d MPI_USE_OMP=1
            -pnode -pre myhook.sh
            myapp")
```

Listing 8. ARC MPI-Start example

```
Arguments="-t openmpi -d MPI_USE_OMP=1
           -pnode -pre myhook.sh
           myapp";
```

Listing 9. gLite MPI-Start example

```
Arguments:
    { mpi-type: openmpi,
      pre: myhook.sh,
      Per node: 1,
      MPI-Start Variable: MPI_USE_OMP=1,
    },
```

Listing 10. UNICORE MPI-Start example

4 INFRASTRUCTURE MONITORING

The execution of parallel application does not only require the middleware support for such jobs, it also needs a correct configuration of the infrastructure where the jobs are actually run. Grid infrastructures are mainly used for the execution of collections of sequential jobs [14], hence the support for parallel applications was not a priority. However, the infrastructure is composed of clusters where execution of parallel applications is possible. In order to assure the correct execution of these applications and, therefore, attract more users to the infrastructure, monitoring probes that check the proper support for such jobs has been introduced.

The monitoring probes are executed at all the sites that announce the support for MPI-Start and they consist in the following steps:

1. Assure that MPI-Start is actually available.
2. Check of the information published by the site. This first step inspects the announced MPI flavors supports and selects the probes that will be run in the next steps.
3. For each of the supported MPI flavors, submit a job to the site requesting 2 processes that is compiled from source using the MPI-Start hooks. The probe checks that the number of processes used by the application was really the requested number.

Although the probes request a low number of slots (2), the existence of such probes allows, both to infrastructure operators and users, to easily detect problems. These probes are flagged as critical, thus any failure may cause the site to be suspended from the infrastructure. The introduction of these probes over the last year has improved the quality of the MPI support significantly thanks to the commitment of the site administrators to ensure no failures in the tests.

5 CONCLUSIONS

The execution of parallel applications in grid environments is a challenging problem that requires the cooperation of several middleware tools and services. The support from middleware is constantly improving and the three computing middleware stacks of EMI provide ways to execute MPI jobs. However, the support varies from one implementation to other and users still need to care about too many details. With the use of MPI-Start, users do not need to worry about all the low level aspects of starting MPI applications in a heterogeneous infrastructure such as the grid. The latest developments in MPI-Start have introduced better control of job execution and the integration with ARC, gLite and UNICORE. Users are totally abstracted by using the unique interface of MPI-Start and can easily migrate their application from one middleware provider to another.

The EGI Infrastructure is committed to the support of parallel applications and provides monitoring probes that allow early detection of any problems that may arise at the sites. The capability of executing MPI jobs and having a single interface for all kind of resources and MPI implementations creates an attractive infrastructure for users from different scientific communities with specific computational needs. The usage of parallel applications will arise with the availability of multiple core machines, the latest developments of MPI-Start provide a better control for the location and number of processes and will continue to improve those features in future releases. The use of advanced topologies, FPGAs, GPGPUs, and massively multi-node jobs will be investigated for use on high-end resource types.

REFERENCES

[1] Aiftimiei, C. et al.: Design and Implementation of the gLite CREAM Job Management Service. Future Generation Computer Systems, Vol. 26, 2010, pp. 654–667.

[2] Anjomshoaa, A. et al.: Job Submission Description Language (JSDL) Specification, Version 1.0. GFD-R.056, 2005.

[3] Bayucan, A.—Henderson, R. L.—Lesiak, C.—Mann, B.—Proet, T.—Tweten, D.: Portable Batch System: External Reference Specification. Technical report, MRJ Technology Solutions, 1999.

[4] Dichev, K. et al.: MPI Support on the Grid. Computing and Informatics, Vol. 27, 2008, pp. 213–223.

[5] Ellert, M. et al.: Advanced Resource Connector Middleware for Lightweight Computational Grids. Future Generation Computer Systems, Vol. 23, 2007, pp. 219–240.

[6] Erwin, D: UNICORE – A Grid Computing Environment. Lecture Notes in Computer Science, Vol. 2150, 2001, pp. 825–834.

[7] European Grid Initiative (EGI) web site. Available on: `http://www.egi.eu/`.

[8] European Middleware Initiative (EMI) web site. Available on: `http://www.eu-emi.eu/`.

[9] Extended Resource Specification Language. NORDUGRID-MANUAL-4, 2011.

[10] Gabriel, E. et al.: Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. Lecture Notes in Computer Science, Vol. 3241, 2004, pp. 97–104.

[11] Gentzsch, W.: Sun Grid Engine: Towards Creating a Compute Power Grid. In Proceedings of the first IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001, pp. 35–36.

[12] Gropp, W.—Lusk, E.—Doss, N.—Skjellum, A.: A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. Parallel Computing, Vol. 22, 1996, No. 6, pp. 789–828.

[13] Gropp, W.: MPICH2: A New Start for MPI Implementations. In Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, 2002, pp. 7.

[14] Iosup, A. et al.: The Grid Workloads Archive. Future Generation Computer Systems, Vol. 24, 2009, No. 7, pp. 672–686.

[15] Karonis, N. T. et al.: MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. J. Parallel Distrib. Comput, Vol. 63, 2003, No. 5, pp. 551–563.

[16] Keller, R.—Gabriel, E.—Krammer, B.—Müller, M. S.—Resch, M. M.: Towards Efficient Execution of MPI Applications on the Grid: Porting and Optimization Issues. Journal of Grid Computing, Vol. 1, 2003, No. 2, pp. 133–149.

[17] Laure, E. et al.: Programming the Grid Using gLite. EGEE-PUB-2006-029, 2006.

[18] Litzkow, M.—Livny, M.—Mutka, M.: Condor – A Hunter of Idle Workstations. In Proceedings of the 8th International Conference of Distributed Computing Systems, 1988, pp. 104–111.

[19] Pacini, F.—Maraschini, A.: Job Description Language (JDL) Attributes Specification. Technical Report 590869, EGEE Consortium, 2006.

[20] Squyres, J. M.: A Component Architecture for LAM/MPI. In Proceedings of the ninth ACM SIGPLAN symposium on principles and practice of parallel programming, 2003, pp. 379–387.

[21] Yoo, A.—Jette, M.—Grondona, M.: SLURM: Simple Linux Utility for Resource Management. In Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, Vol. 2862, 2003, pp. 44–60.

[22] Zhou, S: Lsf: Load Sharing in Large-Scale Heterogeneous Distributed Systems. In Proceedings of the Workshop on Cluster Computing, 2002.

Enol Fernández-del-Castillo joined the Advanced Computing and e-Science group at Instituto de Fsica de Cantabria in Santander (Spain) in 2009. He received his B. Sc. in Computer Engineering in 2003 from the Universidad de La Laguna (Spain) and his Ph. D. in 2008 from the Universidad Autonoma de Barcelona (Spain). He has participated in several Spanish and international projects in the distributed computing area, including CrossGrid, int.eu.grid, Euforia, and EGEE projects. He has developed tools for interactive and parallel computing in grid environments and is the main developer of the CrossBroker resource management system. Currently he is involved in EMI project developing tools for parallel jobs and in the EGI-InsPIRE project in the software provisioning tasks.