Description and Packaging of MPI Applications for Automatic Deployment on Computational Grids

Sébastien Lacour, Christian Pérez, Thierry Priol

I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
N° 1721

DESCRIPTION AND PACKAGING OF MPI APPLICATIONS FOR AUTOMATIC DEPLOYMENT ON COMPUTATIONAL GRIDS

SÉBASTIEN LACOUR, CHRISTIAN PÉREZ, THIERRY PRIOL

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

Description and Packaging of MPI Applications for Automatic Deployment on Computational Grids

Sébastien Lacour, Christian Pérez, Thierry Priol*

Systèmes numériques
Projet Paris

Publication interne n1721 — May 24th, 2005 — 17 pages

Abstract: Computational grids promise to deliver a vast computer power as transparently as the electric power grid supplies electricity. Thus, applications need to be automatically and transparently deployed on grids, in particular MPI parallel applications. However, deploying such applications on a grid is complex: the user must select compute nodes manually, launch processes on distributed, heterogeneous nodes, and configure the MPI library so it can adapt to its grid environment. Our objective is to hide the complexity of application deployment on computational grids. This paper proposes a software architecture designed to automatically deploy MPI applications on a grid. It also introduces a description and packaging model of MPI applications. Finally, the paper shows how the MPI library can be automatically configured with network topology information to optimize collective operations.

Key-words: Automatic Application Deployment, MPI, Computational Grids, Application Description, Network Topology.

(Résumé : tsvp)

* {Sebastien.Lacour,Christian.Perez,Thierry.Priol}@irisa.fr

Centre National de la Recherche Scientifique
(UMR 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

Description et packaging d'applications MPI en vue de leur déploiement automatique sur des grilles de calcul

Résumé : L'un des objectifs des grilles de calcul est de fournir de la puissance informatique de manière aussi transparente que le réseau de distribution de l'électricité fournit de la puissance électrique. Ainsi, les applications doivent se déployer sur les grilles de calcul automatiquement et de façon transparente, en particulier en ce qui concerne les applications parallèles MPI. Cependant, le déploiement de telles applications dans un environnement de grille est une tâche complexe : l'utilisateur doit sélectionner manuellement les nœuds de calcul, lancer des processus sur des nœuds distribués et hétérogènes, puis configurer la bibliothèque MPI pour qu'elle puisse s'adapter à son environnement d'exécution. Notre objectif est de cacher la complexité du déploiement d'applications sur des grilles de calcul. Cet article propose une architecture logicielle conçue pour déployer automatiquement des applications MPI sur des grilles. Il présente également un modèle de description et de packaging d'applications MPI. Enfin, cet article montre comment la bibliothèque MPI peut être automatiquement configurée, en y injectant la description de la topologie réseau, afin d'optimiser la performance des opérations collectives MPI.

Mots clés : Déploiement automatique d'applications, MPI, grilles de calcul, description d'application, topologie réseau.

1 Introduction

Computational grids have the potential to offer a vast computer power. One of the long term goals of computational grids is to provide computer power in the same way as the electric power grid supplies electricity, *i.e.* transparently. Here, *transparency* means that the user does not know what particular resources provide computer power. So the user should just have to submit his or her application to a computational grid and get back the result of the application without worrying about resource selection, resource location and type, or mapping processes on resources. In other words, application deployment should be as *automatic* and easy as plugging an electric appliance into an electric outlet, without requiring expert knowledge in computational grids.

MPI applications are an important class of applications which may be run on a grid, as shown by the number of projects developing grid-enabled MPI implementations (MPICH-G2 [8], MagPIe [10], PACX-MPI [9], *etc.*) A typical grid infrastructure hosting an MPI application is composed of a federation of geographically distributed sites: each site is made of several clusters interconnected by a local-area network, and each cluster is made of a number of compute nodes interconnected by a high-performance system-area network. In order to improve performance, grid-enabled MPI implementations can often take advantage of the knowledge of the underlying network topology in case the application is executing on a hierarchical network infrastructure.

However, application deployment is a complex task in a grid environment, because of network and computer heterogeneity (network topology and performance characteristics, compatibility of operating systems and computer architectures, CPU counts and speeds, *etc.*), security enforcement, resource distribution, variety of job submission methods, *etc.* In addition, the runtime configuration of grid-enabled MPI libraries makes it even more difficult: MPICH-G2, MagPIe, and PACX-MPI need to know the underlying network topology of their execution environment to improve their performance.

Therefore, it is necessary to *hide all that complexity from the user's view* by automating the deployment process of applications on computational grids. This paper proposes extensions to MPI to allow for automatic deployment of MPI applications. To this end, the paper introduces an MPI application description model, a packaging model, and a configuration process. Those elements are integrated into a software architecture which supports scheduling heuristics as plug-ins to build up a deployment tool. This paper also presents ADAGE [1], our prototype deployment tool.

This paper is organized as follows. Section 2 sums up a few specifics of running MPI applications on computational grids, and introduces grid-enabled MPI implementations. Then, Section 3 presents the state of the art in application deployment to show that there is room for improvement as far as automatic deployment of MPI applications is concerned. Section 4 presents our proposed specialization of the general process of automatic deployment for MPI applications, and an implementation is presented in Section 5 through our prototype deployment tool. Section 6 concludes the paper and presents perspectives.

```
cluster 0
process 0 1 2 3

cluster 1
process 4 5 6 7
```

Figure 1: Example of MagPIe configuration file describing the underlying topology of MPI processes: it specifies two clusters of 4 processes each.

2 MPI Applications and Grids

Running certain MPI applications on a computational grid makes sense, depending on their computation and communication patterns: this section mentions how MPI implementations are usually improved and adapted to a grid environment. Then it shows that the targeted usage transparency of computational grids has not been reached yet: MPI application deployment is still a complex task requiring too much manual intervention of the user.

2.1 MPI Collective Operation Optimization

Network hierarchy of computational grids is an important issue which MPICH-G2 [8], MagPIe [10], and PACX-MPI [9] address: grids are made of various sites interconnected over intercontinental or national wide-area networks (WAN), those sites are composed of clusters interconnected by a local-area network (LAN), and clusters are made of computers equipped with a high-performance system-area network (SAN, like Myrinet, InfiniBand, *etc.*) This network topology results in several performance gaps between each interconnection level in terms of latency and bandwidth.

MPI collective operations are widely used: their performance characteristics are a critical question while choosing an MPI library, as exemplified by the efforts spent on the MPICH test suite [6] or SKaMPI [25] to determine performance evaluation methods of collective operations. As they involve a number of geographically distributed processes, implementing MPI collectives in a network topology-aware manner yields significant improvements [7, 10] on grids.

To that end, MPICH-G2, MagPIe, and PACX-MPI try to minimize communications over slower links. This means that the MPI runtime library must be aware of the underlying topology of interconnection between all the processes. In MPICH-G2, this is done using environment variables which the user must define prior to launching MPI processes. In MagPIe and PACX-MPI, a network topology description file (Figure 1) may be provided by the user and installed where the MPI application is to run.

As wide-area network performance has significantly increased over recent years, running a parallel application on a computational grid makes more and more sense, provided

```
(&(resourceManagerContact="clstr.site1.org")
  (count=4)
  (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
    (GLOBUS_LAN_ID Univ_of_Stanford)
    (LD_LIBRARY_PATH "/usr/globus/lib"))
  (executable="/h/user/my_MPI_app_i386")
  (directory="/h/user/"))
(&(resourceManagerContact="clstr.site2.org")
  (count=8)
  (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
    (GLOBUS_LAN_ID Univ_of_Stanford)
    (LD_LIBRARY_PATH "/ap/globus2.4.3/lib"))
  (executable="/h/john/MPI_appl_sparc"))
```

Figure 2: Example of RSL script to launch an MPICH-G2 application (simplified).

that the application has a reasonable communication to computation ratio. Examples of applications which have been successfully run on a grid are given in [2, 15] (more than 1000 nodes for the last one). It makes all the more sense as grid-enabled MPI implementations can take into account the performance gaps between every level of grid network hierarchy.

2.2 Direct Access to Underlying Network Topology

MPI programmers are not only interested in using topology-aware MPI collective operations: they may also wish to access the underlying topology of interconnection between MPI processes in order to optimize their own numerical algorithms. For instance, an algorithm may rely on groups of processes with intense communications within a group, and sparser communications between groups. Then, it would be natural to create groups of intensively communicating processes within the same cluster, and avoid groups made of processes distributed over continents with poor network connection performance characteristics.

As an example, MPICH-G2 provides the MPI programmer with an MPI-compliant mechanism to access the underlying network topology [8]: this allows a process to determine in what cluster and what local-area network it is running, as well as what other processes are part of the same cluster or LAN.

2.3 Grid-Enabled MPI Implementations

Some MPI implementations have been adapted to run on computational grids: MPICH, MagPIe and PACX-MPI may be coupled with the Globus Toolkit to make it easier to deploy

MPI applications on general purpose, shared grid resources. The Globus Toolkit allows for file staging (including executables) and remote process creation.

For instance, to launch an MPI application with MPICH-G2, the user must supply an RSL2 script (Globus2 Resource Specification Language). As illustrated on Figure 2, the user must *manually specify* in this file *where* the application is to run, *how many* processes must be launched on each cluster, as well as the path to the right executable for each cluster. Note that the user must also manually specify that both clusters are on the same local-area network using the environment variable `GLOBUS_LAN_ID`.

Thus, the issues of grid resource management (with the help of the Globus Toolkit) and topology-aware programming algorithms (as seen in previous subsections) have already been addressed in grid-enabled MPI implementations — at least partially. However, resource selection, process placement, compiled executable selection, and automatic configuration of the MPI runtime library have not been tackled yet as far as MPI applications on grids are concerned.

This paper aims to fill this gap by making a step toward more transparency in MPI application deployment on a computational grid using an automatic deployment tool.

3 State of the Art in Application Deployment

To keep the promise of computational grids, *i.e.* delivering a vast computer power transparently, it is important to be able to deploy applications *automatically*. Many types of applications are expected to be run on grids. This paper focuses on MPI applications and assumes a static application: the support of `MPI_Comm_spawn` of the MPI-2 standard, which dynamically launches MPI processes, is out of the scope of this paper as it requires on-line scheduling heuristics. It is a perspective of this work.

This section starts by presenting the state of the art in application deployment on grids, as well as our software architecture for automatically deploying *static* applications. Then it presents three missing elements for automatic deployment of MPI applications. The rest of this paper will propose solutions to those issues.

3.1 Deployment of Applications

Some work has already been done in the field of deployment of certain types of applications. The Configuration Description, Deployment, and Lifecycle Management working group of the Global Grid Forum works on the deployment of web services on grids [3]. This work focuses on service description and configuration, as well as deployment lifecycle management. However, it assumes that nodes have already been pre-allocated before deployment.

The Model Driven Architecture (MDA) and the CORBA Component Model (CCM) of the Object Management Group (OMG) offer a broader vision of “deployment”. The MDA deployment and configuration specification [18] includes the following steps in the deployment process: application description and packaging, deployment planning to select

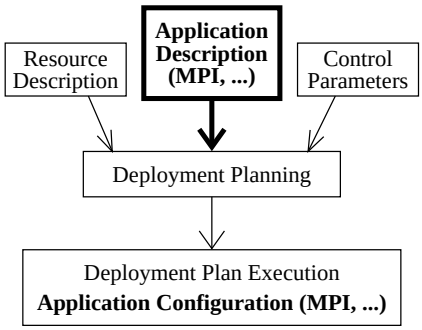

Figure 3: Overview of the general process of automatic application deployment. The focus of this paper is figured in bold.

the distributed resources which will run the application, file installation (staging input data files and executables), process launch and application configuration.

We share MDA's vision of deployment, since we consider compute nodes are not pre-allocated: the resources of a grid are shared by users and not dedicated to a particular application. So the planning step belongs to deployment.

3.2 Automatic Deployment Architecture Overview

In [13], we have proposed a general process to *automatically* deploy applications on a computational grid. The proposed architecture is based on the analysis that automatic, transparent, and efficient deployment of applications on grids requires to fully decouple the *application* description from the *resource* description. Indeed, applications are seldom bound to specific resources, and may be deployed in any environment, like grids. It is the automatic deployment tool's responsibility to discover and select grid resources for execution, and map the processes of the application on the selected resources. The appropriateness of the deployment tool's decisions depends on the quality of the information it is provided with, especially application description.

Figure 3 presents the major steps of our architecture for automatic deployment.

Resource Description. A first input of a deployment tool is a description of the available grid resources. It includes the description of storage and compute resources (CPU counts and speeds, operating systems, computer architectures, *etc.*) which is now well-mastered as exemplified by the information service of the Globus Toolkit (MDS2 [5]). In addition, grid resource description should also include information on the network interconnections between compute nodes, like topology and performance characteristics [14]. That infor-

```
a) size=4 or size=8
b) size>4 and size<32 and even(size)
c) square(size) and size>16
d) grid(3,4) # 4 neighbors
e) gridDiag(3,4) # 8 neighbors
f) grid(size,2*size) and
   square(size) and size>=4
g) grid(3,4,5) and priority(2,0,1)
h) grid(3,4) and bandwidth(1,10)
i) grid(3,4) and latency(5,1)
```

Figure 4: Examples of MPI application descriptions constraining the number of processes (a-c) or the Cartesian grid topology (d-i); (g-i) introduce examples of keywords to order the dimension priority according to some properties.

mation is very important as the application description may impose communication constraints on the execution resources [12].

Application Description. The application description contains information about the various processes to deploy. It should only deal with concepts belonging to the programming model of the application, and should not be bound to specific grid resources. Moreover, it must include pointers to the application's compiled implementations (executables), as well as information on their target operating systems and computer architectures, dependencies on libraries, *etc.* The application description may also express constraints like communication requirements between any two processes. Section 4.1 proposes an MPI application description model. CCM component-based application description is a good source of inspiration: CCM application packages are self-contained and their description includes all the necessary information for automatic deployment.

Control Parameters. Control parameters are additional requirements which the user may impose to keep a certain level of control on the automatic deployment process. For instance, the user may wish to minimize the execution time by selecting computers with the highest CPU speeds available, or run the application at a particular site close to a visualization node. Control parameters should not be part of the application description because they deal with *how* the application should be executed.

Deployment Planning. As shown on Figure 3, the deployment planning phase is at the heart of automatic application deployment: it consists in selecting compute nodes, and mapping the application processes onto the selected compute nodes. It is also responsible for selecting the application's executables to upload onto each computer, and determining the job submission methods to use in order to launch processes. To achieve these tasks,

the planning phase relies on some off-line scheduling algorithms [22] like Condor's Matchmaking mechanism [21]. Their inputs are a description of the application and a description of the available resources, and they produce a deployment plan. The goal of this paper is not to propose a scheduling heuristic, but to present an architecture where scheduling algorithms can be plugged in.

The control parameters supported may vary depending on the quality of the deployment planner. They may also control the selection of a scheduling algorithm depending on the scheduling criteria to optimize.

Grid applications like those based on MPICH-G2, MagPIe, and PACX-MPI may run with a variable number of processes, so the process count may not be fully determined by the application description: the scheduling heuristics [4, 16] should be able to set the number of processors to allocate to an MPI application. As MPI applications may have various communication patterns between their processes, MPI application descriptions may also include network related constraints: the scheduling heuristics [16] should support them too.

Deployment Plan Execution. A deployment planner produces a deployment plan which specifies what actions the automatic deployment tool must perform. Conceptually, this step can be subdivided into three phases. First, various files (executables, input data, *etc.*) are uploaded and installed on the selected compute nodes using the file transfer method specified by the deployment plan (*e.g.*, SCP, GridFTP, *etc.*) Second, processes are launched using the specified remote process creation method (*e.g.*, SSH, Globus, *etc.*) Third, the application is configured: for instance, a grid-enabled MPI application needs to have its network topology information set up. The first two steps are independent of the application, whereas the last phase is application specific.

3.3 Summary and Discussion

Deployment of web services or components on computational grids has received much attention recently. For example, we have successfully validated this general process with the automatic deployment of component-based applications and with a simple planner [11].

However, little effort has been made on the deployment of MPI applications, while they are an important class of grid applications as shown in Section 2.

To allow for automatic deployment of MPI applications, we have identified three main issues which need to be tackled: MPI application description, deployment planning, and configuration of the application during deployment plan execution. Those three issues are addressed in the following sections.

4 Automatically Deploying MPI Applications

This section shows how we propose to specialize the general steps of the automatic deployment process (Section 3.2) to MPI applications. First, it presents a description and

packaging model of MPI applications. Then, it deals with MPI-specific planning issues, and discusses MPI runtime library configuration.

4.1 MPI Application Description

To our knowledge, there is no description model of MPI applications available. All the information used when actually launching an MPI application is held by the user. He or she knows whether an application should run with at least two processes, with an even number of processes, with a square number of processes, *etc.* Thus, we propose a basic topology description model for "flat" MPI applications, and a hierarchical topology description model which is based on the basic description model.

Basic Topology Description. A first issue is to specify a model to describe MPI applications. This paper does not claim to solve this issue completely. As MPI only handles processes, we propose to describe an MPI application using constraints related to the number of processes. For example, it should be possible to declare an exact number of processes, a range (*i.e.*, minimum and maximum numbers), or constraints like an even number, a power-of-two number, *etc.* Logical operators (and, or, not, greater than, less than, equal to, parentheses, *etc.*) should of course be allowed.

These constraints lead to view the processes as an unstructured collection of processes. MPI also allows for the description of two types of virtual topologies: Cartesian grids and graphs. Thus, an MPI application description should allow to attach constraints on these topologies. While constraints related to Cartesian grids are quite intuitive (number of dimensions, constraints on the number of elements for each dimension, *etc.*, see Figure 4), constraints for graphs are not so widely used, so they are not addressed in this paper.

However, as explained in [23], Cartesian grid topologies need to be associated with more information to be really useful. First, it should be specified whether the diagonal neighbor processes should be taken into account (case e. in Figure 4). Second, the dimension needs to be ordered according to a priority (case g.), and / or to the relative importance of bandwidth (in case h., communication along dimension 2 requires 10 times as much bandwidth as along dimension 1) and / or latency (case i.)

Hierarchical Topology Description. While the description of the classical topology of an MPI group seems adequate for "flat" MPI applications, it may not be sufficient for topology-aware grid-enabled MPI applications. A grid-enabled MPI application expects to manage groups of MPI processes with different inter- and intra-group communication performance characteristics. Hence, the application description should allow for such topological constraints. As a consequence, the description of the application may also include group related constraints, like the number of groups, the size of a group, *etc.* For example, an application may impose a maximum of 4 groups, each group being made of a power-of-two number of processes. Figure 5 presents two examples with unstructured groups (case

```
a) group.count=2 and # 2 groups
   group1.size=4 and group2.size=8
b) group.size=16 and group.count=2 and
   group1.size>=4 and group2.size<=8
c) group.size=16 and group.count=2 and
   group1.size>=4 and group1.grid(2, n)
d) group.size=16 and group.count=3 and
   bandwidth(group1,group2)=10 and
   bandwidth(group2,group3)=5
```

Figure 5: Examples of MPI application descriptions constraining process groups. In case d) bandwidth between group1 and group3 does not matter (it is not constrained).

```
Application type: MPICH-G2
Process count: 32
Implementation 1:
   OS: Linux
   ISA: i386
   Location: appl.exe
Implementation 2:
   OS: Solaris
   ISA: Sparc
   Location:
       http://mpi.org/FFT.exe
```

Figure 6: Example of MPI application description.

a. and b.), an example with a structured group (case c.), and an example of communication constraints between process groups (d.)

It is important to note that such a description does not impose a unique physical topology for the selected compute nodes. For instance, constraint b) of Figure 5 may lead to select a single cluster of 16 nodes and create 4 groups of 4 processes each. In that case, the physical machine is a cluster with a uniform network. But it can also lead to select two clusters of 8 nodes each. The application is expected to run efficiently in all those situations provided that the MPI runtime library is topology-aware.

4.2 MPI Application Packaging

Several standard technologies can be used to package an MPI application, like a ZIP archive. The interesting point is the definition of a file format to describe the contents of the archive. For example, the *Open Software Description* (OSD) is an XML vocabulary [24] used by the

OMG to describe CORBA components. As it supports various compiled versions of an application, it is possible to embed several executables for different computers and operating systems. Thus, an application package is self-described: it includes all the necessary information for resource selection and automatic deployment.

4.3 Deployment Planning of MPI Applications

MPI planning algorithms have two related issues to handle. First, they must decide how many processors to actually allocate to the application. Second, they must map a virtual topology to a physical infrastructure. While a good scheduling algorithm should handle those two points, previous works only address the first one. As far as processor count is concerned, there are few theoretical works [4, 16] which appear very interesting as they model meta-tasks whose actual number of tasks (*i.e.*, processes for MPI applications) is decided algorithmically.

Previous works [17, 19] have dealt with the mapping of MPI virtual topologies to physical infrastructures: [19] proposes two algorithms to map virtual topologies to (homogeneous) multi-level clusters; the first algorithm optimizes the total communication cost, and the second one balances the communication load. [19] deals with switch-based clusters with irregular topologies and general virtual topologies.

It is important to support several scheduling heuristics so as to let the user select his or her criteria for a *particular* execution. Hence, an interface should be defined between a deployment tool and scheduling heuristics. This interface seems straightforward: the inputs of the scheduling algorithm are the application description (Section 4.1), the resource description, and the control parameters; the output is a deployment plan (Section 3.2).

4.4 Network Topology Information Management

The last phase of deployment plan execution consists in configuring the application, and in particular providing the MPI runtime library with topology information. Indeed, as introduced in Section 2, grid-enabled MPI implementations need to be aware of the underlying network topology. As this piece of information is already held by the planning algorithm, it can be transmitted to the MPI library by the deployment tool.

As the various grid-enabled MPI implementations use different methods to configure their runtime library, the deployment tool must adapt to a number of MPI implementations to support them. Thus, the application description must specify which MPI implementation is used. Ideally, an application should only express that it depends on MPI, not on a particular MPI *implementation*. As long as a configuration interface for MPI applications has not been standardized (or an ABI, *Application Binary Interface*), either a grid-enabled MPI application is specific to a particular MPI implementation, or it cannot use specific features to discover its underlying network topology.

4.5 Discussion

Three steps of the general process described in Section 3 have been specialized to support automatic deployment of MPI applications. They mainly concern the description and packaging of an MPI application, and the configuration of the MPI runtime library with the underlying network topology information. Hence, it is possible to automatically deploy MPI applications. The following section introduces ADAGE, our research prototype deployment tool, which is able to deploy MPICH-G2 applications.

5 Automatic Deployment of MPI Applications Using ADAGE

This section details the validation of our proposed automatic deployment software architecture within our prototype deployment tool ADAGE (Automatic Deployment of Applications in a Grid Environment, [1]).

5.1 Overview of ADAGE

ADAGE complies with the general software architecture described in Section 3, and includes the specialized features for MPI applications presented in Section 4. However, ADAGE is not restricted to MPI applications: it can also deploy component-based applications as illustrated in [11].

In input, ADAGE needs a packaged MPI application, the description of the available grid resources, and optional control parameters. These descriptions are represented as XML documents (DOM). Hence, the interface between ADAGE and the scheduling heuristics are those DOM documents in input, and a DOM document describing the deployment plan in output. Currently, we have implemented two basic scheduling algorithms (round-robin and random) which respect simple control parameters (*e.g.*, enable Globus or not) as well as operating system and computer architecture compatibility between compute nodes and the application's compiled executables. The deployment planner also selects a job submission method (SSH, or Globus2) based on its availability and control parameters.

Finally, ADAGE executes the deployment plan: it stages files in to the selected computers using SSH or GridFTP (executables, input data files, *etc.*) and launches remote processes. It also configures the application by setting environment variables: currently, ADAGE works with the grid-enabled MPI implementation MPICH-G2 only.

5.2 Automatic Deployment of MPICH-G2 Applications

The unique command to deploy an MPI application on a grid with ADAGE is:

```
adage_deploy -res my_grid.xml -ctrl ctrl_params.xml -appl my_MPI_appl.zip
```

The file `my_grid.xml` is the description of the available grid resources, complying with [12]: they include compute node information (computer architectures, operating systems, *etc.*) as well as network information (topology and performance characteristics). This file is written

```
<job_partitioning type="split">
   <min_size>8</min_size>
</job_partitioning>
```

Figure 7: Example of control parameter specifying a minimum number of processes per cluster.

just once for a given computational grid, and may be re-used to deploy other applications (should they be MPI or not).

The file **`ctrl_params.xml`** contains a set of control parameters expressed in a simple XML vocabulary (attribute name / value). Currently, it allows to specify such requirements as a minimum and / or a maximum number of MPI processes per cluster for instance, as illustrated on Figure 7.

The file **`my_MPI_appl.zip`** is the self-described application package. A schematic example of its contents is illustrated on Figure 6. In this example, the MPICH-G2 application must be run on 32 processes. Two compiled versions of the application are available (Linux / PC and Solaris / Sparc).

Then, one of the two simple heuristics is called (round-robin by default). It discovers grid resources by traversing the resource description files. It respects simple application constraints (compatibility between computers' and executables' operating systems and architectures), as well as control parameters constraints (like MPI process group sizes).

After uploading executables to the selected compute nodes, ADAGE launches the application processes. ADAGE takes care of setting the configuration parameters of the application, so that the MPICH-G2 library can retrieve the underlying network topology: this is done by defining MPICH-G2 specific environment variables.

6 Conclusion

The success of computational grids partially depends on their ease of use. Ultimately, an application should simply be submitted to a grid service which is responsible for finding and selecting grid resources for the application, transferring executables, and actually launching the application. This paper makes a step in that direction by proposing a software architecture to automate deployment of (possibly grid-enabled) MPI applications on grids.

This paper proposes a model to describe and package standard and grid-enabled MPI applications with topology properties. It also shows how planners (*i.e.*, scheduling algorithms) can be plugged in the deployment tool. Then, the application configuration phase, which is crucial for performance, can be transparently handled by the deployment tool to correctly configure the MPI runtime library with network topology information.

This paper has presented ADAGE, our research prototype for automatic deployment of applications: it can deploy MPICH-G2 applications automatically. Deploying an MPI

application on a heterogeneous, distributed grid with ADAGE is as *simple* as deploying it on a homogeneous cluster. Indeed, the ADAGE command, as shown in Section 5.2, is very similar to the `mpirun` command. Neither does the user need to write an RSL file, nor does he have to be aware that the MPI runtime library needs to be configured. Hence, ADAGE shows that *automatic deployment* of MPI applications is feasible.

Though this paper targets grids, this work can also be applied to automatically and transparently deploy MPI applications on parallel systems, like clusters of clusters. Hence, application developers do not have to be aware of the actual cluster configuration, and cluster administrators can upgrade their cluster without disturbing users. The key point is to decouple the application description from the resources.

In order to support both the volatility of grid resources and the dynamicity of some applications (like MPI-2 and the `MPI_Comm_spawn` function), the deployment software architecture needs to be revised. We think this revision is a refinement of the static deployment model.

While executing MPI applications on a grid is important, we are in fact interested in embedding (possibly grid-enabled) MPI applications into parallel components such as GRID-CCM [20]. One of our goals is to efficiently support code coupling applications. As a consequence, the deployment tool needs to deploy applications made of sequential and parallel components, some of them being MPI-based components. Hence, we are working on a unique model to describe both component-based and MPI applications, as well as a deployment planner which would be able to deal with this parallel component model.

References

[1] ADAGE. `http://www.irisa.fr/paris/ADAGE/`, URL.

[2] G. Allen, T. Dramlitsch, I. Foster, N.T. Karonis, M. Ripeanu, E. Seidel, and B. Toonen. Supporting efficient execution in heterogeneous distributed computing environments with Cactus and Globus. In *Proc. of the SuperComputing Conf.*, pages 52–76, Denver, CO, November 2001.

[3] D. Bell, T. Kojo, P. Goldsack, S. Loughran, D. Milojicic, S. Schaefer, J. Tatemura, and P. Toft. Configuration description, deployment, and lifecycle management. CDDLM working group of the global grid forum (GGF), January 2003.

[4] J. Blazewicz, M.Y. Kovalyov, M. Machowiak, D. Trystram, and J. Weglarz. Scheduling malleable tasks on parallel processors to minimize the makespan. *Annals of Operations Research*, 129(Issue 1–4):65–80, July 2004.

[5] Karl Czajkowski, Steven Fitzgerald, Ian Foster, and Carl Kesselman. Grid information services for distributed resource sharing. In *10th Int'l Symp. on High-Performance Distributed Computing*, pages 181–194, San Francisco, CA, August 2001.

[6] Bronis de Supinski and Nick Karonis. Accurately measuring MPI broadcasts in a computational grid. In *8th Symp. on High Performance Distributed Computing (HPDC)*, pages 29–37, Redondo Beach, CA, August 1999.

[7] Nicholas Karonis, Bronis de Supinski, Ian Foster, William Gropp, Ewing Lusk, and John Bresnahan. Exploiting hierarchy in parallel computer networks to optimize collective operation performance. In *14th Int'l Parallel and Distributed Processing Symp. (IPDPS)*, pages 377–384, May 2000.

[8] Nicholas T. Karonis, Brian Toonen, and Ian Foster. MPICH-G2: a grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing (JPDC)*, 63(5):551–563, 2003.

[9] Rainer Keller, Bettina Krammer, Matthias S. Mueller, Michael M. Resch, and Edgar Gabriel. MPI development tools and applications for the grid. In *Workshop on Grid Applications and Programming Tools*, Seattle, WA, June 2003.

[10] T. Kielmann, H. E. Bal, S. Gorlatch, K. Verstoep, and R. F. H. Hofman. Network performance-aware collective communication for clustered wide area systems. *Journal of Parallel Computing*, 27(11):1431–1456, 2001.

[11] Sébastien Lacour, Christian Pérez, and Thierry Priol. Deploying CORBA components on a computational grid: General principles and early experiments using the Globus Toolkit. In *2nd Int'l Working Conf. on Component Deployment (CD 2004)*, number 3083 in LNCS, pages 35–49, Edinburgh, Scotland, UK, May 2004. Springer-Verlag.

[12] Sébastien Lacour, Christian Pérez, and Thierry Priol. A network topology description model for grid application deployment. In *5th Int'l Workshop on Grid Computing (GRID 2004)*, pages 61–68, Pittsburgh, PA, November 2004.

[13] Sébastien Lacour, Christian Pérez, and Thierry Priol. A software architecture for automatic deployment of CORBA components using grid technologies. In *1st Francophone Conf. On Software Deployment and (Re)Configuration*, pages 187–192, Grenoble, France, October 2004.

[14] Bruce Lowekamp, Brian Tierney, Les Cottrell, Richard Hughes-Jones, Thilo Kielmann, and Martin Swany. A hierarchy of network performance characteristics for grid applications and services. Proposed Recommendation Global Grid Forum (GGF), Network Measurement Working Group (NMWG), January 2004.

[15] G. Mahinthakumar, F. M. Hoffman, W. W. Hargrove, and N. Karonis. Multivariate geographic clustering in a metacomputing environment using Globus. In *Proc. of the SuperComputing Conf.*, pages 5–16, Portland, OR, 1999.

[16] Loris Marchal, Yang Yang, Henri Casanova, and Yves Robert. A realistic network / application model for scheduling divisible loads on large-scale platforms. In *Int'l Parallel and Distributed Processing Symposium (IPDPS)*, April 2005.

[17] S. Moh, C. Yu, D. Han, H. Y. Youn, and B. Lee. Mapping strategies for switch-based cluster systems of irregular topology. In *8th Int'l Conf. on Parallel and Distributed Systems*, pages 733–740, Kyongju City, Korea, June 2001.

[18] OMG. Deployment and configuration of component-based distributed applications specification. Draft Adopted Specification ptc/03-07-02, June 2003.

[19] J.M. Orduña, F. Silla, and J. Duato. A new task mapping technique for communication-aware scheduling strategies. In *30th Int'l Workshops on Parallel Processing (ICPP)*, pages 349–354, Valencia, Spain, September 2001.

[20] Christian Pérez, Thierry Priol, and André Ribes. A parallel CORBA component model for numerical code coupling. *The Int'l Journal of High Performance Computing Applications (IJHPCA)*, 17(4):417–429, 2003.

[21] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *7th Int'l Symp. on High Performance Distributed Computing*, pages 140–146, Chicago, IL, July 1998.

[22] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. In *8th Heterogeneous Computing Workshop*, pages 15–29, April 1999.

[23] J. L. Träff. Implementing the MPI process topology mechanism. In *Proc. of the SuperComputing Conf.*, pages 1–14, Baltimore, MD, 2002.

[24] A. van Hoff, H. Partovi, and T. Thai. The open software description format (OSD). `http://www.w3.org/TR/NOTE-OSD.html`, August 1997. Submitted to the W3C.

[25] Thomas Worsch, Ralf Reussner, and Werner Augustin. On benchmarking collective MPI operations. In *9th European PVM/MPI Users' Group Meeting*, number 2474 in LNCS, pages 271–279, Linz, Austria, September 2002. Springer-Verlag.