MPI support in the DIRAC Pilot Job Workload Management System

A.Tsaregorodtsev, V.Hamar
Centre de Physiques de Particules de Marseille (CPPM). France
E-mail: {atsareg,hamar}@in2p3.fr

Abstract: Parallel job execution in the grid environment using MPI technology presents a number of challenges for the sites providing this support. Multiple flavors of the MPI libraries, shared working directories required by certain applications, special settings for the batch systems make the MPI support difficult for the site managers. On the other hand the workload management systems with Pilot Jobs became ubiquitous although the support for the MPI applications in the Pilot frameworks was not available. This support was recently added in the DIRAC Project in the context of the GISELA Latin American Grid Initiative. Special services for dynamic allocation of virtual computer pools on the grid sites were developed in order to deploy MPI rings corresponding to the requirements of the jobs in the central task queue of the DIRAC Workload Management System. Pilot Jobs using user space file system techniques install the required MPI software automatically. The same technique is used to emulate shared working directories for the parallel MPI processes. This makes it possible to execute MPI jobs even on the sites not supporting them officially. Reusing so constructed MPI rings for execution of a series of parallel jobs increases dramatically their efficiency and turnaround. In this contribution we describe the design and implementation of the DIRAC MPI Service as well as its support for various types of MPI libraries. Advantages of coupling the MPI support with the Pilot frameworks are outlined and examples of usage with real applications are presented.

Introduction
Running parallel applications using Grid resources represents a challenge because of diverse environments and complicated requirements such as: shared home directories, specific libraries or compilers. Meeting these requirements is a hard task for grid site managers.

Parallel applications based on the MPI technology [1] are heavily used in grid environments especially in domains other than High Energy Physics. For example, in the GISELA Latin American Grid project parallel applications make up to 30% of the total [2]. However, these applications meet most of the problems: a low number of sites supporting MPI; the MPI software installation procedure is not standardized, as a result different MPI versions can be encountered; some sites install special packages to start MPI applications and the jobs must have special wrappers in order to be executed in these environments. Distribution of input/output files depends also on the site MPI installation: some sites use shared home file systems and others allow secure shell connections between the MPI nodes. Even if a site selected to run an MPI job meets all the requirements, there is no guarantee that the job will be executed successfully. That makes it hard for the user to select the sites where the jobs can be safely executed.

On the other hand, the DIRAC Workload Management Systems (WMS) using pilot jobs is now widely used [3], although the support for MPI applications in the pilot frameworks was not available until recently. This support was added in the DIRAC Project in the context of the GISELA Grid. A special service for dynamic allocation of virtual computer pools on the Grid sites was developed in order to deploy MPI rings corresponding to the requirements of user jobs in the central task queue of the
DIRAC WMS. MPI DIRAC extension makes it possible to execute MPI jobs even in the sites that are not supporting this mode of operation officially. In order to accomplish this task, MPI pilot jobs are in charge of:

- MPI software installation in the Worker Node (WN);
- mounting remote disk on a DIRAC application server containing the required application software as a user space file system;
- starting a file system server on the master MPI node to emulate a shared working directory between the WNs than belong to the same MPI ring in order to allow access to input files during job execution.

In this paper parallel applications and MPI standards are discussed in Section 1, DIRAC middleware support for MPI applications is detailed in Section 2. Results of the test with real applications are described in Section 3 followed by Conclusions and outline of the future work.

1. Parallel applications and MPI

Nowadays, a lot of applications need multiple computers simultaneously in order to be executed in a reasonable time. To achieve that, parallel programming techniques are used based on message-passing mechanisms to communicate and synchronize different subtask between the different processes.

MPI (Message-Passing Interface) forum was created in order to establish a standard interface for message passing implementations and became the de facto standard. MPI is not a language; it is a set of operations expressed as functions, subroutines, or methods, according to the appropriate language bindings.

There exist two main MPI specifications: MPI-1 and MPI-2 [1]. The principal difference between them is that MPI-1 model has no shared memory concept, although MPI-2 has only a limited distributed shared memory concept. There are multiple implementations of the MPI specification, or MPI flavors. Each one consists of a specific set of routines directly callable from C, C++, Fortran, Python, or Java. Examples of MPI implementations are: MPICH, MPICH2, OpenMPI, LAM. A brief description of some implementations is given below:

- MPICH is a portable implementation of MPI-1 specification; this is free software available for most flavors of Unix [4].
- MPICH2 is a high-performance and widely portable implementation of MPI-1 and MPI-2 standards [5]. It is one of the most popular implementations of MPI. MPICH2 replaces MPICH1 and should be used instead of MPICH1 except for the case of clusters with heterogeneous data representations; MPICH2 does not yet support those systems.
- OpenMPI Project is an open source MPI-2 implementation, combining technologies and resources from several other projects (FT-MPI, LA-MPI, LAM/MPI, and PACX-MPI) [6].

In the next section a brief description of the MPI support in DIRAC will be presented. The MPI flavors supported by the DIRAC MPI Agent are: MPICH and MPICH2. Taking into account the vast variety of the MPI implementations, the design of the MPI Agent is made such that adding support for a new MPI flavor can be done without making changes to the agent main code.

2. MPI support in DIRAC

DIRAC uses Pilot Jobs (or Pilots) to create dynamically a distributed Workload Management System [7]. The Pilots are reserving the computing resource, the Worker Node (WN), check the WN environment and then contact the central Task Queue to request a suitable user job. In conjunction with the MPI support, the main idea is that the MPI Pilots must be capable to create a suitable environment on the WNs to execute MPI jobs, even if the sites are not providing MPI support. Besides providing support for common flavors of MPI, the possibility of adding new flavors in the future must be considered. The DIRAC MPI Service components are detailed below (figure 1).
The MPI Service orchestrates the MPI Pilot Jobs behavior, determines the actual ring status depending on the information collected from all the MPI Pilot Jobs belonging to a particular ring and provides this information on the client requests. This information can contain the role assigned to a MPI Pilot Job (master or slave), ring status, etc. The MPI Service state is kept in the MPI Job Database.

The MPI Pilot Job is responsible to start the MPI Agent in the WN, calculate the remaining time in the queue and decide if a new cycle can be started or if the MPI Pilot Job execution must finalize.

The MPI Agent is in charge of establishing contact with the MPI Service; sending the information about the WN environment; creating, starting, finalizing the MPI rings according to the information received from MPI Service.

The MPI Job Database stores the information about all the MPI rings created and the associated user job keeping track of their statuses. It stores also information about the currently active MPI Pilot Jobs including their roles and the Pilot ID complementing the DIRAC WMS Pilot Job Database.

It is important to note that the MPI Pilot Jobs are fully complying with the security standards of the EGI grid being actually standard gLite jobs.

2.1. Pilot roles

To execute MPI user jobs in the grid it is necessary to accumulate the required number of MPI Pilot Jobs in one site and in one network domain. According to the current grid practices, one MPI Pilot Job is associated to only one CPU core.

To create an MPI ring, each MPI Pilot Job can take one of the two roles: master or slave, where only one master can be present in one MPI ring. The first MPI Pilot who makes a request to the MPI Service will start a new ring provided that there are Waiting MPI user job in the Task Queue. The Pilot will take the role of the master of the ring. At this moment the status of the MPI user job is changed to Matched. The MPI ring construction starts and the ring status changed from Empty to Accumulating. The next MPI Pilots running on the same site and sending requests to the MPI Service are assigned to the MPI ring under the construction. They will get the slave role. This process will continue until the required number of CPUs is accumulated. At this point the MPI ring status is changed to Starting. Now the user application can be executed.

The MPI ring will be destroyed when:
- the MPI job is executed successfully;
- an error occurs during the job execution;
- the required number of slaves is not achieved in a given time period; in those cases the MPI user job is rescheduled.
After the ring is destroyed, the MPI Pilot checks if the available execution time is enough to start a new cycle and sends a new request to the MPI Service if this is the case. Otherwise it will be stopped and the WN is freed.

Figure 2. MPI construction and reuse

To illustrate better this idea, here is a brief explanation of the example shown in figure 2. Suppose that two MPI user jobs, requiring 3 CPUs each, are waiting in the Task Queue. MPI Pilots arrive to a site and the creation of the first ring starts, when the number of required MPI pilots is reached the ring status changes to Starting. When a new MPI Pilot arrives to the same site, it will take again the role master for another MPI user job and construction of the second ring starts. It means that in the same site there can be more than one MPI ring executing without interference with each other.

During this process, a third MPI user job requiring 6 CPUs arrives. When one of the MPI user jobs finalizes, the released MPI Pilot evaluates the time left. Supposing that all the MPI Pilots have enough time left to start a new cycle, one of them will be the first to make a request to the MPI Service and will get the role master of the new ring. The remaining MPI Pilots will become slaves in this case. That means that the MPI Pilot Job role can change each time a new cycle is started.

The process will be repeated as long as the time left allows starting new cycles and there are MPI user jobs waiting in the Task Queue. If one of these conditions is not met, the MPI Pilot Jobs finalize and the resource is freed.

2.2. MPI ring life cycle

The MPI ring state machine is described in the following as also illustrated in figure 3:

Figure 3. MPI ring life cycle.

Empty: this status is achieved when the MPI Pilot is for the first time deployed in a site and there are no other rings in the Accumulating status. It will change to Accumulating after the MPI user job is matched and this Pilot will take the role of master.
**Accumulating:** in this case at least the *master* is deployed in the site, the MPI Pilots arriving at this moment are considered as *slaves* until the required number of reserved CPUs is achieved. After that, the ring status changes to *Starting*. In the case the time for accumulating the MPI Pilots exceeds a predefined limit, the ring status is changed to *Out*.

**Starting:** the master starts the MPI environment depending on the MPI flavor and the status changes to *Ready*. In case of MPICH2, the *mpd* daemon is started. The MPI application software repository is mounted to access the DIRAC application server. A temporary file system server is created in the *master* WN to be used as a shared directory for all the nodes in the ring. It is mounted under /$TMP/$JobID directory in all the nodes to share the input and output files.

**RingInit:** the *slaves* start the MPI environment depending on the MPI flavor required. The ring is completely created and tested. The MPI application software repository is mounted from the DIRAC application server, as well as the local file system created by the *master* under /$TMP/$JobId (see subsection 3.3 below). The ring status changes to *Ready*.

**Ready:** the user job wrapper is started. As a next step, the *master* starts the execution of matched MPI job, and the ring status changes to *Running*.

**Running:** this state is valid until the job finishes the execution or fails. If it finishes without problems the ring status is changed to *Done*, in the other case it is changed to *Failed*.

**Done / Failed:** the MPI Environment is reset, all the daemons of the ring are stopped. The ring status changes then to *Out*. The *master* MPI Pilot stops the temporary file systems. All the Pilots, *masters* and *slaves*, remove all the temporary directories.

**Out:** the Pilot roles are reset; all the MPI Pilots check that the daemons are stopped completely. The MPI Pilot is now ready to start a new cycle, if the remaining CPU time is enough to execute another job. In the opposite case the MPI Pilot finishes execution.

As explained in this subsection an MPI Pilot can participate to the execution of more than one user job. This significantly reduces the load on the workload management systems and prevents wasting of resources due to unnecessary time needed to create additional MPI rings.

2.3. **User space shared file systems**

In the first version of the MPI DIRAC extension, the use of shared file system was not implemented. The application software was distributed during the job execution using SSH copy to transfer files between the WNs. Installation of the application in each WN was thus required. This approach needs a lot of time and does not solve a problem faced by most of the MPI applications: the use of shared files during the job execution. Most of the sites are not supporting shared file systems between the WNs. The possibility to have a single source for the application software is also very attractive as it ensures the same version of the application in all the sites. Also according to general DIRAC rules, the intervention of grid administrators to install the application software must be not required in any case. Therefore, there was a need for a shared file system both across sites and among the WNs of the same MPI ring.

The following technologies to mount network file systems were considered:

- **CERNVM-FS** is used for several Worldwide LHC Computing Grid (WLCG) [8] experiments to distribute specific software for a Virtual Organization (VO) [9]. The CERNVM-FS is implemented as a File System in User Space (FUSE) module [10], and allows creating and showing the directory tree stored as a read only file system on a web server accessible through the HTTP protocol. Advantages of this are: the transfers are done file by file on demand together with the contents verification using SHA1 keys; a network of HTTP caching servers is used to reduce the network latency. In this case, a CERNVM-FS server can be installed and populated with DIRAC applications easily, but the CERNVM-FS client must be installed or available in all the WNs and configured to allow mounting DIRAC applications repositories. For the latter, root level access is required, eliminating CERNVM-FS as the solution until network file systems can be mounted without intervention of the grid site administrators.

- **AFS** also needs the intervention of grid administrators to install and configure clients; this solution has the same problem as the previous option.

- **Cooperative Computing Tools (ctools)** is a package composed by a set of tools [11]. It contains:
  - *Chirp* is a distributed file system designed for exporting access to file data for use in grid computations [12].
  - *Parrot* is a tool that allows mounting local user space file systems communicating to remote repositories with the *Chirp* protocol and other remote access protocols like http, grow, ftp, irods,
hdfs, CERNVM-FS without root or other special privileges. Another advantage is the use of ACLs in each remote directory and the use of GSI standard grid authentication method.

Based on their properties well suited for the needs expressed in this paper, the pair of Chirp and Parrot tools was selected. The chosen solution is illustrated in figure 4.

![Figure 4](image)

**Figure 4.** Filesystems used by the MPI DIRAC extension.

A Chirp server was installed to distribute the applications between all the WNs. ACLs were configured to use the Globus ad authentication method; the server was started and populated with the software for Abinit and SPECFEM3D applications [13].

The MPI Agent was modified in order to install cctools in the WNs as part of the environment verification. The environment variables were changed to point to the DIRAC application server. In the case of MPICH2 MPI flavor, the MPI mpd daemons are started using Parrot commands. As for the local shared file system, the mount points must be defined on the fly to avoid conflicts with other MPI user jobs running in the same site. Strict rules are applied for defining locations of the applications repository and input data files mount points.

The MPI user job script skeleton was modified also to use Parrot commands and to find the input files in the local Chirp server using environment variables predefined by the MPI Agent.

No change to the MPI Service was needed to run with this new solution.

The solution presented here differs from others in that following the DIRAC middleware philosophy, MPI user jobs can be executed in any site, even if MPI is not supported officially, without any intervention of grid site administrators.

3. Tests with real MPI applications

Abinit was chosen among other GISELA applications because it requires to have a shared home directory for all the MPI processes; it uses the MPICH2 MPI flavor. The installation of the application software requires much space, which makes it worth installing outside the WNs user space.

3.1. Abinit

Abinit is a package whose main program allows one to find the total energy, charge density and electronic structure of systems made of electrons and nuclei (molecules and periodic solids) within Density Functional Theory (DFT), using pseudopotentials and a planewave or wavelet basis [13].

To use the Abinit parallel binaries it is required to compile MPICH-2 package with Fortran’90 support. Fortran’90 is not installed in most of the sites, and the problem of versions appears again. In order to avoid this difficulty, gcc binaries are also deployed by MPI Pilot Jobs.
A simple JDL was prepared; the executable in this case is a script that gets the environment variables set by the MPI Agent before the MPI user job execution. It copies also the input files to the temporary file system. After the job execution, the output files are copied from the temporary file system to the grid job working directory to be transferred to the DIRAC Sandbox service or to be copied to the SEs as requested by the job owner.

This application was run successfully in rings of 4 CPUs in multiple GISELA sites.

3.2. Problems encountered

During the preparation and execution of the application some problems were encountered, those are briefly described here.

At the beginning of the tests with Chirp and Parrot the MPI ring was not working properly, which was reported to the cctools team and the solution was provided by them promptly in the next release.

The idea to deploy MPI packages in a Chirp server together with the application and not in the WNs was tried out. Starting the MPI mpd daemons using Parrot commands was tested, in this case the ring was properly created, the execution of mpdtrace command to test the ring status was done without problems, but the difficulties occurred at the moment when the application was executed. The mpirun communication between different nodes in the rings was not made properly, the application froze. In the test run using just one CPU in the ring the application was executed successfully. After this trial, deployment of the MPI software in the WNs turned out to be mandatory. The problem was reported to cctools team and will be likely solved in the near future.

Abinit needs a file where the actual input file names are specified, as this path is generated dynamically by the MPI Agent, this file must be changed before the application execution, because the application does not recognize environment variables as part of input options.

4. Conclusions and future work

MPI applications represent a significant percentage of the user applications in the GISELA grid. This paper presents a different approach of how to execute this kind of applications in distributed computing environments.

Extending DIRAC to support MPI jobs was relatively easy due to the modular architecture of the DIRAC middleware.

The MPI DIRAC components are flexible enough to execute various applications. It opens a new opportunity in the way to use grid computing resources. However, it can be further improved and extended. For the future we plan to include support for other MPI flavors in the MPI Agents. Porting more applications to this environment will help to prove the usefulness of this solution.

The tests of other combinations of packages to mount network file systems for the DIRAC application server to be used with Parrot will be carried out. Actually cctools package is providing support for CERNVM-FS, one of the first options evaluated. This is a very attractive solution as many application repositories are already hosted in CERNVM-FS servers.

With the Cloud Computing playing progressively more important role, the DIRAC Workload Management System is going to be enhanced to deal with this novel computing resources. For the execution of MPI applications it has potentially multiple advantages. The MPI Pilot Jobs can use more than one CPU core improving the times of MPI ring creation. Other ways of distributing application software can be evaluated for this computing environment.

5. Acknowledgements

The authors would like to thank Dr. Douglas Thain, from University of Notre Dame for his valuable help and support. This work was supported also by the GISELA project, funded by the European Commission under the Grant Agreement # 261487.

References
[1] http://www.mpi-forum.org
[2] http://www.gisela-grid.eu
[3] http://diracgrid.org
[4] http://www.mcs.anl.gov/research/projects/mpi/mpich1-old
[5] http://www.mcs.anl.gov/research/projects/mpich2
[6] OpenMPI project: http://www.open-mpi.org
   Lam project: http://www.lam-mpi.org
   LA-MPI project: http://public.lanl.gov/lampi
   FT-MPI project: http://icl.cs.utk.edu/ftmpi/overview/index.html
   PACX-MPI project: http://www.hlrs.de/organization/av/amt/research/pacx-mpi/
[7] A.Casajus, R.Graciani, A.Tsaregorodtsev, DIRAC pilot framework and the DIRAC Workload Management System, 2010 J. Phys.: Conf. Ser. 219 062049
[8] http://wlcg.web.cern.ch
[9] http://cernvm.cern.ch
[10] http://fuse.sourceforge.net
[11] http://www.nd.edu/~ccl
[12] Douglas Thain, Christopher Moretti, and Jeffrey Hemmes. 2008. Chirp: A Practical Global Filesystem for Cluster and Grid Computing. (Journal of Grid Computing, Springer. DOI: 10.1007/s10723-008-9100-5).
[13] http://www.abinit.org
   http://www.geodynamics.org/cig/software/specfem3d