Consolidation of MPI support of EMI compute elements: a uniform MPI user experience

Enol Fernández

CSIC

Background

"The highest priority is given to providing a common interface to multi-core jobs on all resources"

- Meeting at EGI TF 2010
  - Ivan (ARC), Bernd (UNICORE), Enol (gLite)
  - Put in common how MPI jobs are handled for 3 cases: simple precompiled, simple with compilation and gromacs

MPI on the Grid

- Submission/Allocation
  - Definition of job characteristics
  - Search and select adequate resources
  - Allocate (or coallocate) resources for the job

- Execution
  - File distribution
  - Batch system interaction
  - MPI implementation details

Allocation / Submission (ARC)

```
&(jobName="openmpi-gcc64")
(count="16")
(wallTime="10 minutes") (memory="1024")
(executable="runopenmpi.sh")
(executables="hello-ompi-gcc64.exe" "runopenmpi.sh")
(inputfiles=("hello-ompi-gcc64.exe" ""))
(stdout="std.out")
(stderr="std.err")
(gmlog="gmlog")
(runtimeenvironment="ENV/MPI/OPENMPI-1.3/GCC64”)
```

Allocation / Submission (gLite)

```
Type           = "Job";
CPUNumber     = 16;
Executable     = "mpi-startwrapper.sh";
StdOutput      = "std.out";
StdError       = "std.err";
InputSandBox   = {"my_app", "mpi-startwrapper.sh"};
OutputSandBox = {"std.out", "std.err"};
Requirements     =
  Member("OPENMPI",
       other.GlueHostApplicationSoftwareRunTimeEnvironment)
  && Member("MPI-START",
       other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

WholeNode = True;
NodeNumber = 4;
SMPGranularity = 4;

Allocation / Submission (UNICORE)

```
{
  Executable: "/home/rbreu/mympi/hello_world",

  Resources:{ CPUs: 16, },

  Execution environment: {
    Name: "OpenMPI",
    Arguments: {
      Number of Processes: 16,
    },
  },

}
```

```
Resources:{
 CPUsPerNode: 4,
 Nodes: 4,
}
```

Allocation / Submission (EMI-ES v0.17)

- NumberOfSlots:
  - Total number of slots
- SlotsPerHost (optional):
  - number of slots on each single host.
- ExclusiveExecution (optional):
  - whether a host should be allocated for exclusive use by the job
- ProcessPerHost (optional):
  - number of instances of the executable per host.
- ThreadsPerProcesses (optional):
  - number of threads per process (i.e., per instance of the executable)

Execution

- There is no standard way of starting an MPI application
  - No common syntax for mpirun, mpiexec support optional
- The cluster where the MPI job is supposed to run doesn't have a shared file system
  - How to distribute the binary and input files?
  - How to gather the output?
- Different clusters over the Grid are managed by different Local Resource Management Systems (PBS, LSF, SGE,…)
  - Where is the list of machines that the job can use?
  - What is the correct format for this list?
- How to compile MPI program?
  - How can a physicist working on Windows workstation compile his code for/with an Itanium MPI implementation?

RunTime Environment (ARC)

- Set of software, complete with corresponding setup script (bash) which defines necessary UNIX environment variables, allowing execution of specific applications.
- Three step invocation at the CE:
  - 0: made before the the batch job submission script is written
  - 1: just prior to execution of the user specified executable
  - 2: “clean-up” call, after the user's executable has returned
- User selects the RTE in the job description:
  - (runTimeEnvironment=MYSOFT-v2.0)

Hello World (ARC)

```
&(jobName="openmpi-gcc64")
(count="4")
(wallTime="10 minutes")
(memory="1024")
(executable="runopenmpi.sh")
(executables="hello-ompi.exe" "runopenmpi.sh")
(inputfiles=("hello-ompi.exe" ""))
(stdout="std.out")
(stderr="std.err")
(gmlog="gmlog")
(runtimeenvironment="ENV/MPI/OPENMPI-1.3/GCC64”)
```

```
#!/bin/sh
echo "MPIRUN is '$MPIRUN'”
echo "NSLOTS is '$NSLOTS'”
$MPIRUN -np $NSLOTS ./hello-ompi.exe
```

MPI-Start (gLite)

- Specify a unique interface to the upper layer to run a MPI job.
- Allow the support of new MPI implementations without modifications in the Grid middleware
- Support of “simple” file distribution
- Extensible via pre and post actions (hooks):
  - compilation
  - input preparation
  - management of output
  - hooks have access to MPI-Start internals (e.g. list of hosts)

Hello World (gLite + MPI-Start)

```
JobType          = "Normal";
CpuNumber       = 4;
Executable       = "starter.sh";
InputSandbox     = {"starter.sh", "hello-ompi.exe"}
StdOutput        = "std.out";
StdError         = "std.err";
OutputSandbox    = {"std.out","std.err"};
Requirements     =
    Member("MPI-START”,
           other.GlueHostApplicationSoftwareRunTimeEnvironment)
 && Member("OPENMPI”,
           other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

```
#!/bin/sh

# Set environment variables needed
export I2G_MPI_APPLICATION=hello-ompi.exe
export I2G_MPI_TYPE=openmpi

# Execute mpi-start
$I2G_MPI_START
```

Execution Environment (UNICORE)

- Extension of the JSDL
- Site admin:
  - knows how to configure the available tools
  - adds Execution Env. to UNICORE with the available options.
- User:
  - selects the Execution Env.
  - sets options
  - may customize with pre/post commands

Hello World (UNICORE)

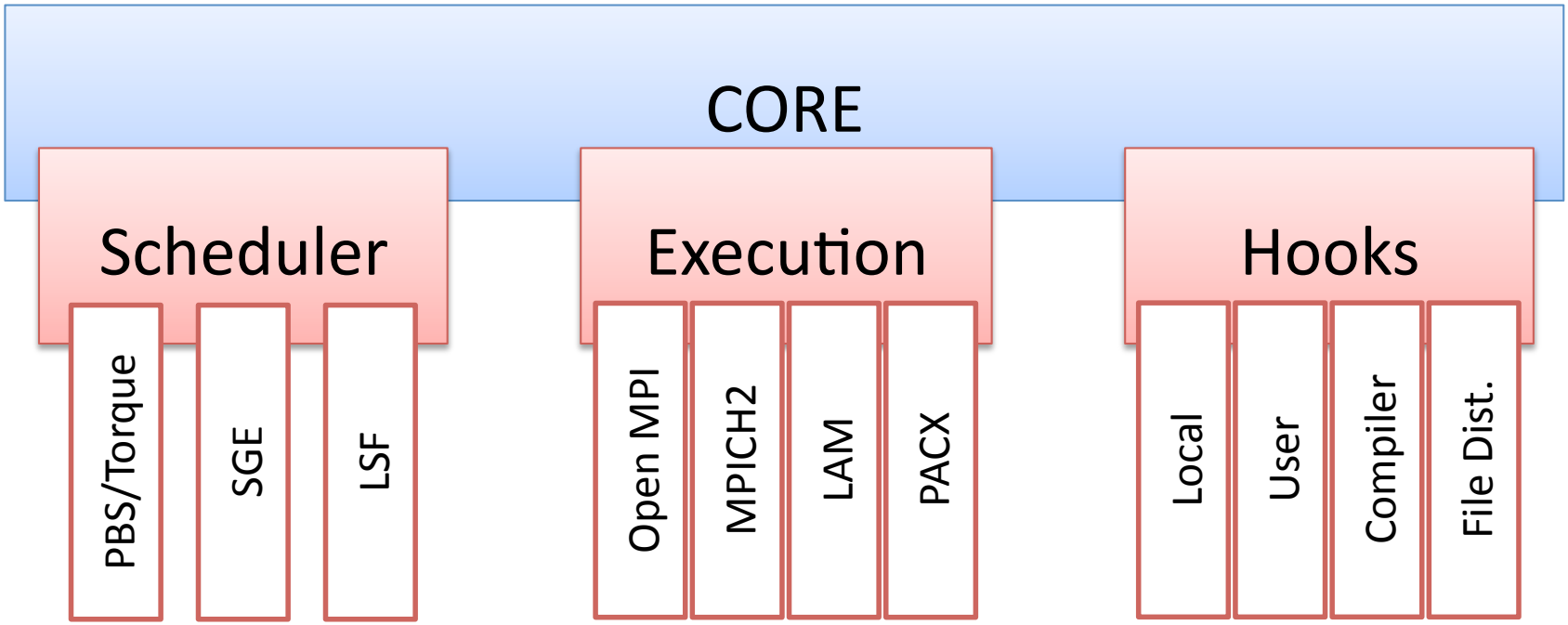
```
{
  Executable: "./hello-ompi.exe",
  Imports: [
    {From: "/myfiles/hello.mpi", To: "hello-ompi.exe" },
  ],
  Resources:{ CPUsPerNode: 2, Nodes: 2, },
  Execution environment: {
    Name: OpenMPI,
    Arguments: { Processes: 4,  },
  },
}
```

No need for user script!

Job execution methods comparison

	RTE	MPI-Start	Execution Env.
“Installation”	Site	Site / User	Site
Invocation	Job Description + user script	Explicit by user	Job Description
What the user needs?	write a script for starting the job using some predefined variables	write a script that sets MPI-Start parameters & invoke MPI-Start	Set Execution Environment parameters at job description
Pre/Post actions	¿User script? Admin can do pre/post actions.	Hooks	Pre / Post commands
Able to change submission script	Yes	No	No

MPI-Start Architecture



MPI-Start

- PROs:
  - single interface for all parallel jobs
  - very easy to deploy for admins without much experience
  - handles common tasks in a uniform way (distribution of files)
  - highly customizable with hooks (both by user & admin)
- CONs:
  - not fully integrated with the site or job submission
  - interface as environment variables may not fit everywhere

MPI-Start + RTE

- Create a new RTE: MPI-Start

```
#!/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=/opt/mpi-start/bin/mpi-start
        export MPI_START_SHARED_HOME=yes
        ;;
2 ) # nothing here
        ;;
* ) # everything else is an error
        return 1
esac
```

MPI-Start + RTE

```
&(jobName="openmpi-gcc64")
(count="4")
(wallTime="10 minutes")
(memory="1024")
(executable="runopenmpi.sh")
(executables="hello-ompi.exe" "runopenmpi.sh")
(inputfiles=("hello-ompi.exe" ""))
(stdout="std.out")
(stderr="std.err")
(gmlog="gmlog")
(runtimeenvironment="ENV/MPI/MPI-START”)
```

```
#!/bin/sh

# Set environment variables needed
export I2G_MPI_APPLICATION=hello-ompi.exe
export I2G_MPI_TYPE=openmpi

# Execute mpi-start
$I2G_MPI_START
```

MPI-Start + Execution Environment

- Execution Environment arguments are used for creation of a single command line
  - MPI-Start arguments are currently set as environment variables
  - MPI-Start would need to change the way it receives options

More Info…

- ARC RTE:
  - http://www.nordugrid.org/applications/environments/
  - Use cases: https://wiki.ndgf.org/display/ndgfwiki/MPI+through+ARC+examples
- MPI-Start:
  - http://devel.ifca.es/mpi-start
  - Use cases: http://devel.ifca.es/mpi-start/wiki/EMIUseCases
- Unicore EE:
  - http://sourceforge.net/apps/mediawiki/unicore/index.php?title=ExecutionEnvironment
  - Use cases: https://twiki.cern.ch/twiki/bin/view/EMI/UnicoreMPIExamples

Status of (gLite) MPI in ETICS

- 3 different componets:
  - mpi-start
  - mpi-yaim
  - gLite-MPI (just a metapackage with mpi-start & mpi-yaim as dependencies)
- All building correctly as part of emi-0

MPI-Start testing

- Until recently, only manual testing of some functionality.
- Now in development:
  - Testing for scheduler modules (detection of number of slots, number of hosts, …)
  - Testing for file distribution (detection of file distribution method, test of the methods)
  - Testing of execution methods