A unified user experience for MPI jobs in EMI

Enol Fernández (CSIC)
gLite MPI PT
Outline

• Parallel Jobs
• EMI middleware stacks approaches
  – How to execute a simple MPI job with 16 process with ARC/glite/UNICORE?
• Unified Job Execution Experience
  – Submission
  – Execution
• MPI-Start in EMI-1
Parallel jobs

Submission/Allocation:
- Definition of job characteristics
- Search and select adequate resources
- Allocate (or co-allocate) resources for job

Execution:
- File distribution
- Batch system interaction
- Parallel job runtime implementation details
Executing an MPI job

• 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?
ARC Parallel Job

& (jobName="openmpi-gcc64")
(count="16")
(wallTime="10 minutes")
(memory="1024")
(executable="runopenmpi.sh")
(executables="hello-ompi.exe" "runopenmpi.sh")
(inputfiles="hello-ompi.exe" "runopenmpi.sh")
(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
gLite Parallel Job

<table>
<thead>
<tr>
<th>JobType</th>
<th>= &quot;Normal&quot;;</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpuNumber</td>
<td>= 16;</td>
</tr>
<tr>
<td>Executable</td>
<td>= &quot;starter.sh&quot;;</td>
</tr>
<tr>
<td>InputSandbox</td>
<td>= {&quot;starter.sh&quot;, &quot;hello-ompi.exe&quot;}</td>
</tr>
<tr>
<td>StdOutput</td>
<td>= &quot;std.out&quot;;</td>
</tr>
<tr>
<td>StdError</td>
<td>= &quot;std.err&quot;;</td>
</tr>
<tr>
<td>OutputSandbox</td>
<td>= {&quot;std.out&quot;,&quot;std.err&quot;};</td>
</tr>
</tbody>
</table>
| Requirements  | = 
  - Member("MPI-START",
    other.GlueHostApplicationSoftwareRunTimeEnvironment)
  &
  - Member("OPENMPI",
    other.GlueHostApplicationSoftwareRunTimeEnvironment); |

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

#!/bin/sh

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

# Execute mpi-start
$I2G_MPI_START
UNICORE Parallel Job

```json
{
    Executable: "./hello-ompi.exe",
    Imports: [
        {From: "/myfiles/hello.mpi", To: "hello-ompi.exe" },
    ],
    Resources:{ CPUs: 16, },
    Execution environment: {
        Name: OpenMPI,
        Arguments: { Processes: 4, },
    },
}
```

No need for user script!
## ARC vs gLite vs UNICORE

<table>
<thead>
<tr>
<th></th>
<th>RTE</th>
<th>MPI-Start</th>
<th>Execution Env.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployment</td>
<td>Site</td>
<td>Site / User</td>
<td>Site</td>
</tr>
<tr>
<td>Invocation</td>
<td>Job Description + user script</td>
<td>Explicit by user</td>
<td>Job Description</td>
</tr>
<tr>
<td>What the user needs?</td>
<td>write a script for starting the job using some predefined variables</td>
<td>write a script that sets MPI-Start parameters &amp; invoke MPI-Start</td>
<td>Set Execution Environment parameters at job description</td>
</tr>
<tr>
<td>Pre/Post actions</td>
<td>User script Admin can do pre/post actions.</td>
<td>Hooks</td>
<td>Pre / Post commands</td>
</tr>
</tbody>
</table>
Unified API for Submission

- EMI-ES (See presentation by Bernd Schuller in this same session)
  - 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)
• Specify a unique interface to the upper layer to run a Parallel (MPI) job
• Allow the support of new MPI implementations without modifications in the Grid middleware
• Portable
• Modular and extensible architecture
• Extensive debugging options
MPI-Start Architecture

Scheduler
- PBS/Torque
- SGE
- LSF

CORE
- Open MPI
- MPICH2
- LAM
- PACX

Hooks
- Local
- User
- Compiler
- File Dist.
• Single interface for all parallel jobs:
  – No need to learn a different command line option every time a MPI version changes
  – No need to learn how each scheduler manages the hosts
• Control of process mapping:
  – One process per host
  – N processes per host
  – K total processes
• File distribution
  – No need to worry (much) about shared or not file systems
• Customizable with hooks
  – Compilation
  – Input preparation
  – Management of output
MPI-Start for site admins

• Single interface for all parallel jobs:
  – No need to configure a different Execution Environment / Run Time Environment for each type of job

• Easy to deploy for admins without much experience:
  – Default MPI installations paths for SL5 (current target for EMI) installations detected
  – Yaim module for configuration

• Customizable with hooks for sites with specific requirements
#!/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 ) # nothing here
       ;;
  *) # everything else is an error
     return 1
esac
<jsdl-u:ExecutionEnvironment>
  <jsdl-u:Name>mpi-start</jsdl-u:Name>
  <jsdl-u:Description>Run parallel applications</jsdl-u:Description>
  <jsdl-u:ExecutableName>/usr/bin/mpi-start</jsdl-u:ExecutableName>
  <jsdl-u:Argument>
    <jsdl-u:Name>mpi type</jsdl-u:Name>
    <jsdl-u:IncarnatedValue>-t</jsdl-u:IncarnatedValue>
    <jsdl-u:ArgumentMetadata>
      <jsdl-u:Description>MPI implementation to use</jsdl-u:Description>
      <jsdl-u:Type>string</jsdl-u:Type>
    </jsdl-u:ArgumentMetadata>
  </jsdl-u:Argument>
  <jsdl-u:Argument>
    <jsdl-u:Name>Per node</jsdl-u:Name>
    <jsdl-u:IncarnatedValue>-npnode</jsdl-u:IncarnatedValue>
    <jsdl-u:ArgumentMetadata>
      <jsdl-u:Description>Number of processes per node</jsdl-u:Description>
      <jsdl-u:Type>int</jsdl-u:Type>
    </jsdl-u:ArgumentMetadata>
  </jsdl-u:Argument>
  ...
</jsdl-u:ExecutionEnvironment>
<jsdl-u:Argument>
  <jsdl-u:Name>Export Environment Variable</jsdl-u:Name>
  <jsdl-u:IncarnatedValue>-x</jsdl-u:IncarnatedValue>
  <jsdl-u:ArgumentMetadata>
    <jsdl-u:Description>Export an environment variable (e.g., "foo=bar" exports the environment variable name "foo" and sets its value to "bar" in the started processes)</jsdl-u:Description>
    <jsdl-u:Type>string</jsdl-u:Type>
  </jsdl-u:ArgumentMetadata>
</jsdl-u:Argument>

<jsdl-u:Argument>
  <jsdl-u:Name>MPI-Start Variable</jsdl-u:Name>
  <jsdl-u:IncarnatedValue>-d</jsdl-u:IncarnatedValue>
  <jsdl-u:ArgumentMetadata>
    <jsdl-u:Description>Define a MPI-Start variable (e.g., "I2G_MPI_START_VERBOSE=1")</jsdl-u:Description>
    <jsdl-u:Type>string</jsdl-u:Type>
  </jsdl-u:ArgumentMetadata>
</jsdl-u:Argument>

<jsdl-u:Argument>
  <jsdl-u:Name>Verbose</jsdl-u:Name>
  <jsdl-u:IncarnatedValue>-v</jsdl-u:IncarnatedValue>
  <jsdl-u:OptionMetadata>
    <jsdl-u:Description>Be verbose</jsdl-u:Description>
  </jsdl-u:OptionMetadata>
</jsdl-u:Argument>
</jsdl-u:ExecutionEnvironment>
& (jobName="mpi-start")
 (count="16")
 (runtimeenvironment="ENV/MPI-START")
 (executable="/usr/bin/mpi-start")
 (arguments="-t openmpi hello-ompi.exe")
 (inputfiles="hello-ompi.exe")
 (stdout="std.out")
 (stderr="std.err")
 (gmlog="gmlog")
 (wallTime="10 minutes")
 (memory="1024")
### gLite Parallel Job Revisited

```plaintext
JobType = "Normal";
CpuNumber = 16;
Executable = "/usr/bin/mpi-start";
Arguments = "-t openmpi hello-ompi.exe";
InputSandbox ={"hello-ompi.exe"}
StdOutput = "std.out";
StdError = "std.err";
OutputSandbox ={"std.out","std.err"};
Requirements =
    Member("MPI-START",
        other.GlueHostApplicationSoftwareRunTimeEnvironment) 
    &&
    Member("OPENMPI",
        other.GlueHostApplicationSoftwareRunTimeEnvironment);
```
UNICORE Parallel Job Revisited

```
{
    Executable: "./hello-ompi.exe",
    Imports: [
        {From: "/myfiles/hello.mpi", To: "hello-ompi.exe" },
    ],
    Resources: { CPUs: 16, },
    Execution environment: {
        Name: mpi-start,
        Arguments: { mpi-type: openmpi, },
    },
}
```
Hybrid OpenMP + MPI

ARC
(arguments="-t openmpi -d MPI_USE_OMP=1 -pnode -pre myhook.sh -post myhook.sh myapp")

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

UNICORE
Arguments: {
    mpi-type: openmpi,
    pre: myhook.sh,
    post: myhook.sh,
    Per node: 1,
    MPI-Start Variable: MPI_USE_OMP=1,
},
pre_run_hook() {
    mpicc -fopenmp ${MPI_MPICC_OPTS} -o myapp myapp.c
    if [ ! $? -eq 0 ]; then
        echo "Error compiling program. Exiting..."
        return 1
    fi
    return 0
}

# the first parameter is the name of a host
my_copy () {
    scp . $1:$PWD/mydata.*
}

post_run_hook () {
    if [ "\$MPI_START_SHARED_FS" = "x0" ]; then
        echo "gather output from remote hosts"
        mpi_start_foreach_host my_copy
    fi
    tar -czvf output.tgz mydata.*
    return 0
}
• MPI-Start 1.0.4 released as part of EMI-1
• Includes:
  – Scheduler plugins for (Sun/Oracle/Univa) GE, PBS/Torque, LSF, Slurm & Condor
  – Execution plugins for Open MPI, MPICH, MPICH2, LAM, PACX-MPI
  – Hooks for OpenMP, Marmot, MPI Trace
• Use of command line parameters (instead of environment variables)
• Updated documentation
• Linux FHS compliant
Thank you

EMI is partially funded by the European Commission under Grant Agreement INFSO-RI-261611