






The EDG Workload Management System

(EDG release 2.0)



Contents

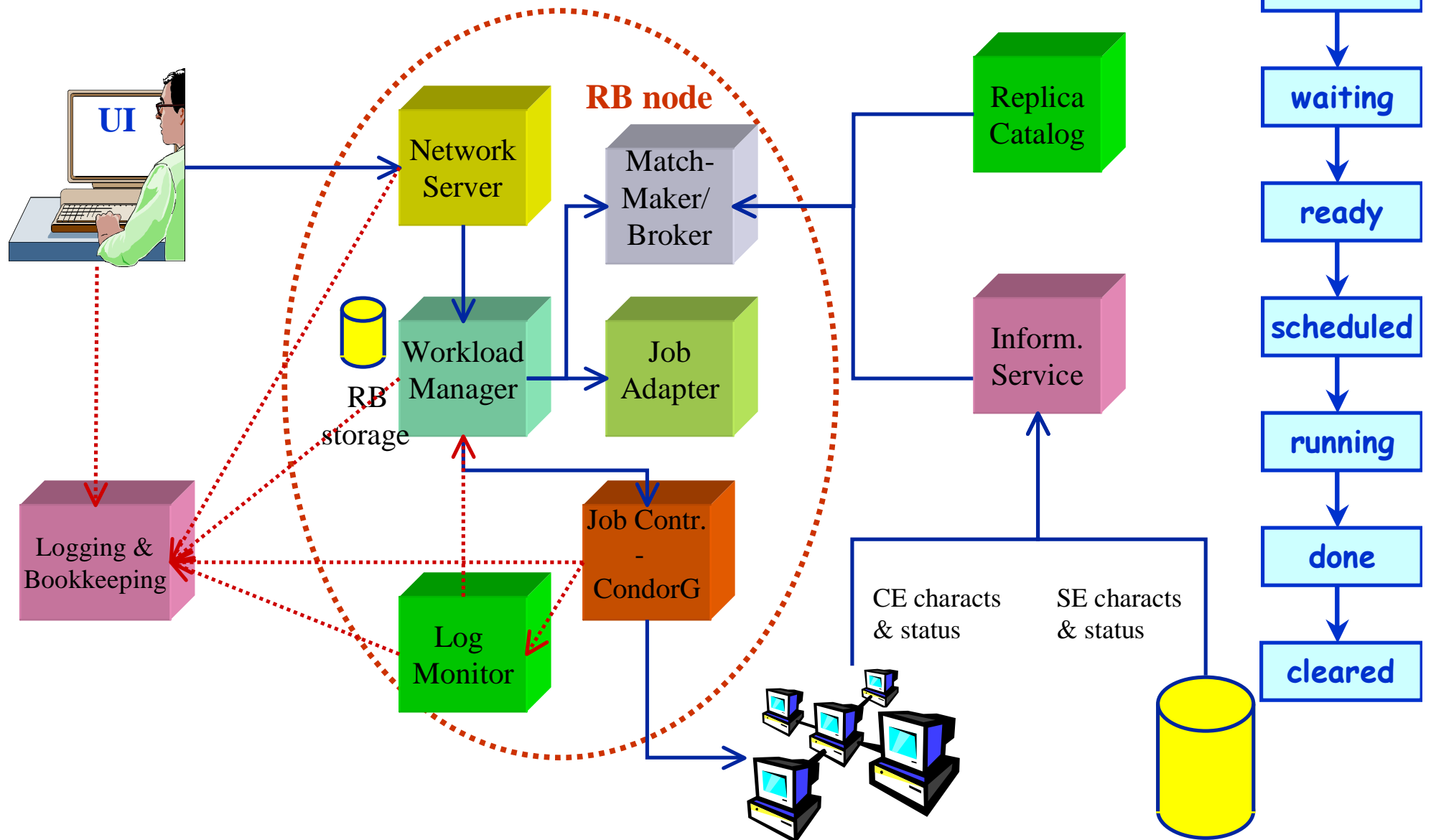
- ◆ Main differences between release 1.4 and 2.0
- ◆ Different job types
 - Normal jobs
 - Interactive jobs 
 - Checkpointable jobs 
 - Parallel jobs 



Main changes to release 1.4

- ◆ WMS re-factored
- ◆ dg-job-* commands changed to **edg-job-***
- ◆ slight JDL changes:
 - Use GLUE schema
 - Interaction with 2.0 data mgmt
- ◆ New features
 - Interactive, checkpointable, MPI jobs
 - Java and C++ API
 - GUI
- ◆ Job state transition simplified
 - 'outputready' state abolished – job should end with 'done (success)'

Job submission





Example of JDL File

```
[  
JobType="Normal";  
Executable = "gridTest";  
StdError = "stderr.log";  
StdOutput = "stdout.log";  
InputSandbox = {"home/joda/test/gridTest"};  
OutputSandbox = {"stderr.log", "stdout.log"};  
InputData = {"lfn:green", "guid:red"};  
DataAccessProtocol = "gridftp";  
Requirements = other.GlueHostOperatingSystemNameOpSys == "LINUX"  
                && other.GlueCEStateFreeCPUs>=4;  
Rank = other.GlueCEPolicyMaxCPUtime;  
]
```



Job Submission

```
edg-job-submit [-r <res_id>] [-c <config file>]  
[-vo <VO>] [-o <output file>] <job.jdl>
```

- r the job is submitted directly to the computing element identified by *<res_id>*
- c the configuration file *<config file>* is pointed by the UI instead of the standard configuration file
- vo the Virtual Organization (if user is not happy with the one specified in the UI configuration file)
- o the generated *edg_jobId* is written in the *<output file>*

Useful for other commands, e.g.:

```
edg-job-status -i <input file> (or edg_jobId)
```

- i the status information about *edg_jobId* contained in the *<input file>* are displayed

Interactive jobs

- ◆ Specified setting `JobType = "Interactive"` in JDL
- ◆ When an interactive job is executed, a window for the stdin, stdout, stderr streams is opened
 - Possibility to send the stdin to the job
 - Possibility to have the stderr and stdout of the job when it is running
- ◆ Possibility to start a window for the standard streams for a previously submitted interactive job with command `edg-job-attach`



Job checkpointing

- ◆ Checkpointing: saving from time to time job state
 - Useful to prevent data loss, due to unexpected failures
 - Approach: provide users with a “trivial” logical job checkpointing service
 - User can save from time to time the state of the job (defined by the application)
 - A job can be restarted from an intermediate (i.e. “previously” saved) job state
- ◆ Different than “classical checkpointing (i.e. saving all the information related to a process: process’s data and stack segments, open files, etc.)”
 - Very difficult to apply (e.g. problems to save the state of open network connections)
 - Not necessary for many applications
- ◆ To submit a checkpointable job
 - Code must be instrumented (see next slides)
 - `JobType=Checkpointable` to be specified in JDL



Job checkpointing example

```
int main ()  
{  
  ...  
  for (int i=event; i < EVMAX; i++)  
    { < process event i>;}  
  ...  
  exit(0); }  
}
```

Example of
Application
(e.g. HEP MonteCarlo
simulation)

Job checkpointing example

```
#include "checkpointing.h"

int main ()
{ JobState state(JobState::job);
  event = state.getIntValue("first_event");
  PFN_of_file_on_SE = state.getStringValue("filename");
  ....
  var_n = state.getBoolValue("var_n");
  < copy file_on_SE locally>;
  ...
  for (int i=event; i < EVMAX; i++)
    { < process event i>;
      ...
      state.saveValue("first_event", i+1);
      < save intermediate file on a SE>;
      state.saveValue("filename", PFN of file_on_SE);
      ...
      state.saveValue("var_n", value_n);
      state.saveState(); }
  ...
  exit(0); }
```

User code
must be easily
instrumented in order
to exploit the
checkpointing
framework ...

Job checkpointing example

```
#include "checkpointing.h"

int main ()
{ JobState state(JobState::job);
  event = state.getIntValue("first_event");
  PFN_of_file_on_SE = state.getStringValue("filename");
  ....
  var_n = state.getBoolValue("var_n");
  < copy file_on_SE locally>;
  ...
  for (int i=event; i < EVMAX; i++)
    { < process event i>;
      ...
      state.saveValue("first_event", i+1);
      < save intermediate file on a SE>;
      state.saveValue("filename", PFN of file_on_SE);
      ...
      state.saveValue("var_n", value_n);
      state.saveState(); }
  ...
  exit(0); }
```

- User defines what is a state
- Defined as <var, value> pairs
- Must be “enough” to restart a computation from a previously saved state

Job checkpointing example

```
#include "checkpointing.h"

int main ()
{ JobState state(JobState::job);
  event = state.getIntValue("first_event");
  PFN_of_file_on_SE = state.getStringValue("filename");
  ....
  var_n = state.getBoolValue("var_n");
  < copy file_on_SE locally>;
  ...
  for (int i=event; i < EVMAX; i++)
    { < process event i>;
      ...
      state.saveValue("first_event", i+1);
      < save intermediate file on a SE>;
      state.saveValue("filename", PFN of file_on_SE);
      ...
      state.saveValue("var_n", value_n);
      state.saveState(); }
  ...
  exit(0); }
```

User can save
from time to time
the state of the job

Job checkpointing example

```
#include "checkpointing.h"

int main ()
{ JobState state(JobState::job);
  event = state.getIntValue("first_event");
  PFN_of_file_on_SE = state.getStringValue("filename");
  ....
  var_n = state.getBoolValue("var_n");
  < copy file_on_SE locally>;
  ...
  for (int i=event; i < EVMAX; i++)
    { < process event i>;
      ...
      state.saveValue("first_event", i+1);
      < save intermediate file on a SE>;
      state.saveValue("filename", PFN of file_on_SE);
      ...
      state.saveValue("var_n", value_n);
      state.saveState(); }
  ...
  exit(0); }
```

Retrieval of the last saved state
The job can restart from that
point



Job checkpointing scenarios

◆ Scenario 1

- Job submitted to a CE
- When job runs it saves from time to time its state
- Job failure, due to a Grid problems (e.g. CE problem)
- Job resubmitted by the WMS possibly to a different CE
- Job restarts its computation from the last saved state
 - → No need to restart from the beginning
 - → The computation done till that moment is not lost

◆ Scenario 2

- Job failure, but not detected by the Grid middleware
- User can retrieve a saved state for the job (typically the last one)
 - *edg-job-get-chkpt -o <state> <edg-jobid>*
- User resubmits the job, specifying that the job must start from a specific (the retrieved one) initial state
 - *edg-job-submit -chkpt <state> <JDL file>*

Submission of parallel jobs

- ◆ Possibility to submit MPI jobs
- ◆ MPICH implementation supported
- ◆ Only parallel jobs inside a single CE can be submitted
- ◆ Submission of parallel jobs very similar to normal jobs
 - Just needed to specify in the JDL:
 - `JobType= "MPICH"`
 - `NodeNumber = n;`
 - The number (n) of requested CPUs
- ◆ Matchmaking
 - CE chosen by RB has to have MPICH sw installed, and at least n total CPUs
 - If there are two or more CEs satisfying all the requirements, the one with the highest number of free CPUs is chosen



Further information

- ◆ The EDG User's Guide

<http://marianne.in2p3.fr>

- ◆ EDG WP1 Web site

<http://www.infn.it/workload-grid>

In particular WMS User & Admin Guide and JDL docs

- ◆ ClassAd

<https://www.cs.wisc.edu/condor/classad>