CREAM$_{G}uideDocumentation$

Release 0.1

Paolo Andreetto

Dec 17, 2019

Contents

Contents:

CHAPTER 1

CREAM Functional Description

The CREAM (Computing Resource Execution And Management) Service is a simple, lightweight service that implements all the operations at the Computing Element (CE) level; its well-defined WebService-based interface and its implementation as an extension of the Java-Axis servlet (running inside the Apache Tomcat container) provide interoperability with clients written in any programming language and running on any computer platform.

The CREAM interface is well-defined using the Web Service Description Language (WSDL); anyone can generate his/her CREAM client by simply filling in the stub code generated by WSDL parser (gSOAP for C/C++, Axis for Java, Perl module for perl).

CREAM functionality:

- Job Submission
  - Possibility of direct staging of input sandbox files
  - gLite JDL compliance (with CREAM-specific extensions)
  - Support for batch and parallel jobs
- Manual and automatic proxy delegation
- Job Cancellation
- Job Info with configurable level of verbosity and filtering based on submission time and/or job status
- Job List
- Job Suspension and Resume
- Job output retrieval
- Job Purge for terminated jobs
- Possibility (for admin) to disable new submissions
- Self limiting CREAM behavior: CREAM is able to protect itself if the load, memory usage, etc. is too high. This happens disabling new job submissions, while the other commands are still allowed
- ARGUS or gJAF based authorization
- Possibility to forward requirements to the batch system

- Integration with APEL accounting systems

CREAM can be used

- by the Workload Management System (WMS), via the ICE (Interface to CREAM Environment) service
- by a generic client, e.g. an end-user willing to directly submit jobs to a CREAM CE. A C++ command line interface (CLI) is available

CREAM service accesses and operates local resource management systems. The current version of the application supports the following management systems:

- TORQUE
- LSF
- SLURM
- HTCondor
- Grid Engine (partially)

Authentication in CREAM is managed via the trustmanager. The Trust Manager is the component responsible for carrying out authentication operations. It is an implementation of the J2EE security specifications. Authentication is based on PKI. Each user (and Grid service) wishing to access CREAM is required to present an X.509 format certificate. These certificates are issued by trusted entities, the Certificate Authorities (CA). The role of a CA is to guarantee the identity of a user. This is achieved by issuing an electronic document (the certificate) that contains the information about the user and is digitally signed by the CA with its private key. An authentication manager, such as the Trust Manager, can verify the user identity by decrypting the hash of the certificate with the CA public key. This ensures that the certificate was issued by that specific CA. The Trust Manager can then access the user data contained in the certificate and verify the user identity.

Authorization in the CREAM CE can be implemented in two different ways (the choice is done at configuration time):

- Authorization with ARGUS
- Authorization with gJAF

Argus is a system meant to render consistent authorization decisions for distributed services (e.g. compute elements, portals). In order to achieve this consistency a number of points must be addressed. First, it must be possible to author and maintain consistent authorization policies. This is handled by the Policy Administration Point (PAP) component in the service. Second, authored policies must be evaluated in a consistent manner, a task performed by the Policy Decision Point (PDP). Finally, the data provided for evaluation against policies must be consistent (in form and definition) and this is done by the Policy Enforcement Point (PEP). Argus is also responsible to manage the Grid user - local user mapping.

gJAF (Grid Java Authorization Framework) provides a way to invoke a chain of policy engines and get a decision result about the authorization of a user. The policy engines are divided in two types, depending on their functionality. They can be plugged into the framework in order to form a chain of policy engines as selected by the administrator in order to let him set up a complete authorization system. A policy engine may be either a PIP or a PDP. PIP collect and verify assertions and capabilities associated with the user, checking her role, group and VO attributes. PDP may use the information retrieved by a PIP to decide whether the user is allowed to perform the requested action, whether further evaluation is needed, or whether the evaluation should be interrupted and the user access denied.

In CREAM CE VO based authorization is supported. In this scenario, implemented via the VOMS PDP, the administrator can specify authorization policies based on the VO the jobs' owners belong to (or on particular VO attributes). When gJAF is used as authorization mechanism, the Grid user - local user mapping is managed via glexec, For what concerns authorization on job operations, by default each user can manage (e.g. cancel, suspend, etc.) only her own jobs. However, the CREAM administrator can define specific super-users who are empowered to manage also jobs submitted by other users.

CHAPTER 2

CREAM installation

2.1 Requirements

The requirements for a basic installation of a CREAM site are:

- Scientific Linux 6, Cent OS version 7 or compatibile distributions.
- The UMD4 extension as described in the UMD guide

2.2 Installation

The basic installation of a CREAM site consists on the following mandatory packages:

- canl-java-tomcat
- cleanup-grid-accounts
- bdii
- dynsched-generic
- glite-ce-cream
- glite-info-provider-service
- globus-gridftp-server-progs
- globus-proxy-utils
- glue-schema
- kill-stale-ftp
- lcg-expiregridmapdir
- mariadb
- sudo

If the authorization framework used is based on glexec the following packages are mandatory:

- glexec
- lcas-lcmaps-gt4-interface
- lcas-plugins-basic
- lcas-plugins-check-executable
- lcas-plugins-voms
- lcmaps-plugins-basic
- lcmaps-plugins-verify-proxy
- lcmaps-plugins-voms

if the authorization framework used is based on Argus the following package is mandatory:

- argus-gsi-pep-callout

The following packages are optional:

- apel-parsers : if the support for APEL accounting is enabled
- glite-lb-logger : if the support for the Logging and Bookkeping is enabled
- info-dynamic-scheduler-lsf : if the batch system used is LSF
- info-dynamic-scheduler-lsf-btools : if the batch system used is LSF
- info-dynamic-scheduler-slurm : if the batch system used is SLURM
- lcg-info-dynamic-scheduler-condor : if the batch system used is HTCondor
- lcg-info-dynamic-scheduler-pbs : if the batch system used is TORQUE

The standard procedure for the installation and configuration of a CREAM site makes use of the puppet framework. The puppet module checks for the presence of the packages described above, and in case they're not available on the system it installs them.

2.3 Example

This section illustrate an example of installation of a CREAM site by means of a standalone execution of puppet. In the following examples the FQDN of the CREAM CE is myhost.mydomain.

The required steps are:

- Installation of the packages for puppet:

  ```
  yum -y install puppet
  ```

- Verification of the puppet environment:

  ```
  facter | grep -E 'hostname|fqdn'
  ```

  If the FQDN of the host is not correctly resolved it is necessary to specify the parameter `creamce::host`

- Installation of the CREAM CE module for puppet:

  ```
  puppet module install infnpd-creamce
  ```

- Creation of the required directories:

```
mkdir -p /etc/puppet/manifests /var/lib/hiera/node /etc/grid-security
```

- Creation of the file `/etc/puppet/manifests/site.pp` with the following definition

```
node 'myhost.mydomain' {
  require creamce
}
```

- Creation of the file `/etc/hiera.yaml` with the following definitions:

```
---
:backends:
  - yaml
:hierarchy:
  - "node/%{fqdn}"
:yaml:
  :datadir: /var/lib/hiera
```

- Creation of the symbolic link

```
ln -s /etc/hiera.yaml /etc/puppet/hiera.yaml
```

- Creation of the CREAM CE description file `/var/lib/hiera/node/myhost.mydomain.yaml`, an example of minimal configuration is:

```
---
creamce::mysql::root_password :      mysqlp@$$w0rd
creamce::creamdb::password :         creamp@$$w0rd
creamce::creamdb::minpriv_password : minp@$$w0rd
apel::db::pass :                     apelp@$$w0rd
creamce::batch_system :              pbs
creamce::use_argus :                 false
creamce::default_pool_size :         10

gridftp::params::certificate :       "/etc/grid-security/hostcert.pem"
gridftp::params::key :               "/etc/grid-security/hostkey.pem"
gridftp::params::port :              2811

creamce::queues :
    long :  { groups : [ dteam, dteamprod ] }
    short : { groups : [ dteamsgm ] }

creamce::vo_table :
    dteam : {
        vo_app_dir : /afs/dteam,
        vo_default_se : storage.pd.infn.it,
        servers : [
                      {
                          server : voms.hellasgrid.gr,
                          port : 15004,
                          dn : /C=GR/O=HellasGrid/OU=hellasgrid.gr/CN=voms.
↪hellasgrid.gr,
                          ca_dn : "/C=GR/O=HellasGrid/OU=Certification␣
↪Authorities/CN=HellasGrid CA 2016"
                      },
                      {
                          server : voms2.hellasgrid.gr,
```

(continues on next page)

(continued from previous page)

```
                          port : 15004,
                          dn : /C=GR/O=HellasGrid/OU=hellasgrid.gr/CN=voms2.
↪hellasgrid.gr,
                          ca_dn : "/C=GR/O=HellasGrid/OU=Certification␣
↪Authorities/CN=HellasGrid CA 2016"
                      }
        ],
        groups : {
            dteam : { fqan : [ "/dteam" ], gid : 9000 },

            dteamsgm : { fqan : [ "/dteam/sgm/ROLE=developer" ], gid : 9001, pub_
↪admin : true },

            dteamprod : { fqan : [ "/dteam/prod/ROLE=developer" ], gid : 9002 }
        },
        users : {
            dteamusr : { first_uid : 6000, fqan : [ "/dteam" ],
                         name_pattern : "%<prefix>s%03<index>d" },

            dteamsgmusr : { first_uid : 6100, fqan : [ "/dteam/sgm/ROLE=developer
↪", "/dteam" ],
                            pool_size : 5, name_pattern : "%<prefix>s%02<index>d"␣
↪},

            dteamprodusr : { first_uid : 6200, fqan : [ "/dteam/prod/
↪ROLE=developer", "/dteam" ],
                             pool_size : 5, name_pattern : "%<prefix>s%02<index>d
↪" }
        }
    }

creamce::hardware_table :
    subcluster001 : {
        ce_cpu_model : XEON,
        ce_cpu_speed : 2500,
        ce_cpu_vendor : Intel,
        ce_cpu_version : 5.1,
        ce_physcpu : 2,
        ce_logcpu : 2,
        ce_minphysmem : 2048,
        ce_minvirtmem : 4096,
        ce_os_family : "linux",
        ce_os_name : "CentOS",
        ce_os_arch : "x86_64",
        ce_os_release : "7.0.1406",
        ce_outboundip : true,
        ce_inboundip : false,
        ce_runtimeenv : [ "tomcat_6_0", "mysql_5_1" ],
        subcluster_tmpdir : /var/tmp/subcluster001,
        subcluster_wntmdir : /var/glite/subcluster001,
        ce_benchmarks : { specfp2000 : 420, specint2000 : 380, hep-spec06 : 780 },
        nodes : [ "node-01.mydomain", "node-02.mydomain", "node-03.mydomain" ]
        # Experimental support to GPUs
        accelerators : {
            acc_device_001 : {
                type : GPU,
                log_acc : 4,
```

(continues on next page)

(continued from previous page)

```
                phys_acc : 2,
                vendor : NVidia,
                model : "Tesla k80",
                version : 4.0,
                clock_speed : 3000,
                memory : 4000
            }
        }
    }

creamce::software_table :
    tomcat_6_0 : {
        name : "tomcat",
        version : "6.0.24",
        license : "ASL 2.0",
        description : "Tomcat is the servlet container"
    }
    mysql_5_1 : {
        name : "mysql",
        version : "5.1.73",
        license : "GPLv2 with exceptions",
        description : "MySQL is a multi-user, multi-threaded SQL database server"
    }

creamce::vo_software_dir : /afs

creamce::se_table :
    storage.pd.infn.it : { mount_dir : "/data/mount", export_dir : "/storage/
↪export",
                           type : Storm, default : true }
    cloud.pd.infn.it : { mount_dir : "/data/mount", export_dir : "/storage/export
↪",
                         type : Dcache }
```

The permissions of the file `/var/lib/hiera/node/myhost.mydomain.yaml` must be set to `600`.

- Deployment of the host private key in `/etc/grid-security/hostkey.pem`
- Deployment of the host certificate in `/etc/grid-security/hostcert.pem`
- Execution of puppet

```
puppet apply /etc/puppet/manifests/site.pp
```

CHAPTER 3

Puppet Configuration

The CREAM CE site can be configured with a puppet module. The module resorts to hiera for handling configuration parameters. Parameters are marked by type; simple types are "string", "integer" and "boolean"; complex types are "list" and "hash".

Hiera can work with different back-ends, like YAML and JSON. For the examples of parameter definitions in the following document the YAML syntax is used. According to YAML specification, the complex types can be declared in two different modes:

- with indentation:

```
# Example of list type
blah::shared_directories :
    - "/mnt/afs"
    - "/media/cvmfs"

# Example of hash type, with nested complex definitions
creamce::queues :
    long :
        groups :
            - dteam
            - dteamprod

    short :
        groups :
            - dteamsgm
```

- with block style:

```
# Example of list type
blah::shared_directories : [ "/mnt/afs", "/media/cvmfs" ]

# Example of hash type, with nested complex definitions
creamce::queues : {
    long : { groups : [ dteam, dteamprod ] },
```

(continues on next page)

(continued from previous page)

```
    short : { groups : [ dteamsgm ] }
}
```

Mixing the two modes is possible but may be misleading; in case it is possible to test the hiera configuration with special tools like yamllint, already present in EPEL distribution.

3.1 CREAM service parameters

These are the basic configuration parameters related to the CREAM service:

- `creamce::batch_system` (string): The installed batch system, *mandatory*, one of "pbs", "slurm", "condor", "lsf"
- `creamce::host` (string): The fully qualified Computing Element host name, default the host name
- `creamce::port` (integer): The tomcat listen port, default 8443
- `creamce::site::name` (string): The human-readable name of the site, default the host name
- `creamce::site::email` (string): The main email contact for the site. The syntax is a coma separated list of email addresses, default undefined
- `creamce::sandbox_path` (string): The directory where the sandbox files are staged on the CREAM CE node, default "/var/cream_sandbox"
- `creamce::delegation::purge_rate` (integer): Specifies in minutes how often the delegation purger has to run, default 10 minutes
- `creamce::lease::time` (integer): The maximum allowed lease time in second. If a client specifies a lease time too big, this value is used instead, default 36000 seconds.
- `creamce::lease::rate` (integer): Specifies in minutes how often the job purger has to run, default 30 minutes

The following parameters enable and configure the internal monitoring system. Whenever a resource metric exceeds the related resource limit the CREAM CE suspends the job submission.

- `creamce::enable_limiter` (boolean): In order to disable the limiter, it is needed to set this parameter value to false and restart the service, default true
- `creamce::limit::load1` (integer): Limiter threshold for the load average (1 minute), default 40
- `creamce::limit::load5` (integer): Limiter threshold for the load average (5 minute), default 40
- `creamce::limit::load15` (integer): Limiter threshold for the load average (15 minute), default 20
- `creamce::limit::memusage` (integer): Limiter threshold for the memory usage, default 95 (percentage)
- `creamce::limit::swapusage` (integer): Limiter threshold for the swap usage, default 95 (percentage)
- `creamce::limit::fdnum` (integer): Limiter threshold for the number of file descriptors, default 500
- `creamce::limit::diskusage` (integer): Limiter threshold for the disk usage, default 95 (percentage)
- `creamce::limit::ftpconn` (integer): Limiter threshold for the number of concurrent ftp connections, default 30
- `creamce::limit::fdtomcat` (integer): Limiter threshold for the number of file descriptors, default 800
- `creamce::limit::activejobs` (integer): Limiter threshold for the number of active jobs, default -1 (unlimited)

- `creamce::limit::pendjobs` (integer): Limiter threshold for the number of pending jobs, default -1 (unlimited)

The following parameters configure the internal job purger mechanism, all the values are expressed in number of days

- `creamce::job::purge_rate` (integer): Specifies in minutes how often the job purger has to run, default 300 minutes.
- `creamce::purge::aborted` (integer): Specifies how often the job purger deletes the aborted jobs, default 10 days
- `creamce::purge::cancel` (integer): Specifies how often the job purger deletes the cancelled jobs, default 10 days
- `creamce::purge::done` (integer): Specifies how often the job purger deletes the executed jobs, default 10 days
- `creamce::purge::failed` (integer): Specifies how often the job purger deletes the failed jobs, default 10 days
- `creamce::purge::register` (integer): Specifies how often the job purger deletes the registered jobs, default 2 days

The following parameters configure the job wrapper:

- `creamce::jw::proxy_retry_wait` (integer): The minimum time interval expressed in seconds, between the first attempt and the second one for retrieving the user delegation proxy, default 60
- `creamce::jw::isb::retry_count` (integer): The maximum number of ISB file transfers that should be tried, default 2
- `creamce::jw::isb::retry_wait` (integer): If during a input sandbox file transfer occurs a failure, the JW retries the operation after a while. The sleep time between the first attempt and the second one is the "initial wait time" (i.e. the wait time between the first attempt and the second one) expressed in seconds. In every next attempt the sleep time is doubled. Default 60 seconds.
- `creamce::jw::osb::retry_count` (integer): The maximum number of ISB file transfers that should be tried, default 2
- `creamce::jw::osb::retry_wait` (integer): If during a output sandbox file transfer occurs a failure, the JW retries the operation after a while. The sleep time between the first attempt and the second one is the "initial wait time" (i.e. the wait time between the first attempt and the second one) expressed in seconds. In every next attempt the sleep time is doubled. Default 300 seconds.

3.2 CREAM Database

The following parameters configure the CREAM back-end:

- `creamce::mysql::root_password` (string): root password for the database administrator, *mandatory*
- `creamce::creamdb::password` (string): The database user password for the main operator, *mandatory*
- `creamce::creamdb::minpriv_password` (string): The database user password for the monitor agent, *mandatory*
- `creamce::mysql::max_active` (integer): The maximum number of active database connections that can be allocated from this pool at the same time, or negative for no limit, default 200
- `creamce::mysql::min_idle` (integer): The minimum number of connections that can remain idle in the pool, without extra ones being created, or zero to create none, default 30

- `creamce::mysql::max_wait` (integer): The maximum number of milliseconds that the pool will wait for a connection to be returned before throwing an exception, or -1 to wait indefinitely, default 10000
- `creamce::mysql::override_options` (hash): see the override option defined in https://forge.puppet.com/puppetlabs/mysql,default

```
{'mysqld' => {'bind-address' => '0.0.0.0', 'max_connections' => "450" }}
```

- `creamce::creamdb::name` (string): The database name for the CREAM service, default "creamdb"
- `creamce::creamdb::user` (string): The database user name with user acting as main operator, default "cream"
- `creamce::creamdb::host` (string): The fully qualified host name for any CE databases, default the host name
- `creamce::creamdb::port` (integer): The mysql listen port for any CE databases, default 3306
- `creamce::creamdb::minpriv_user` (string): The database user name with user acting as monitor agent, default "minprivuser"
- `creamce::delegationdb::name` (string): The database name for the Delegation Service, default "delegationcreamdb"

3.3 BLAH

These are the basic configuration parameters for BLAH:

- `blah::config_file` (string): The path of the main BLAH configuration file, default "/etc/blah.config"
- `blah::logrotate::interval` (integer): The interval in days for log rotation, default 365 days
- `blah::logrotate::size` (string): The size of a log file in MB, default "10M"
- `blah::use_blparser` (boolean): If true it enables the BLParser service otherwise BUpdater/BNotifier is used, default false
- `creamce::blah_timeout` (integer): Represents the maximum time interval in seconds accepted by CREAM for the execution of commands by BLAH, default 300 seconds
- `creamce::job::prefix` (string): The prefix to be used for the BLAH job id, default "cream_"
- `blah::shared_directories` (list): A list of of paths that are shared among batch system head and worker nodes; the empty list is the default value

The following parameters configure BLAH if BNotifier/BUpdater are enabled:

- `blah::bupdater::loop_interval` (integer): The interval in seconds between two BUpdater sessions, default 30 seconds.
- `blah::bupdater::notify_port` (integer): The service port for the BNotifier, default 56554
- `blah::bupdater::logrotate::interval` (integer): The interval in days for log rotation, default 50
- `blah::bupdater::logrotate::size` (string): The size of a log file in MB, default "10M"

The following parameters configure BLAH if the BLParser is enabled:

- `blah::blp::host` (string): The host name for the primary BLParser, *mandatory* if BLParser is used, default undefined
- `blah::blp::port` (integer): The service port for the primary BLParser, default 33333
- `blah::blp::num` (integer): The number of BLParser enabled instances, default 1

- `blah::blp::host1` (string): The host name for the secondary BLParser, default undefined
- `blah::blp::port1` (integer): The service port for the secondary BLParser, default 33334
- `creamce::listener_port` (integer): The port used by CREAM to receive notifications about job status changes sent by the BLParser/JobWrapper, default 49152
- `creamce::blp::retry_delay` (integer): The time interval in seconds between two attempts to contact the BLAH parser, default 60 seconds
- `creamce::blp::retry_count` (integer): Represents the number of attempts to contact the BLAH parser (if it is not reachable) before giving up. If -1 is specified, CREAM will never give up , default 100

3.4 CREAM information system

The following parameters configure the Resource BDII:

- `bdii::params::user` (string): The local user running the BDII service, default "ldap"
- `bdii::params::group` (string): The local group running the BDII service, default "ldap"
- `bdii::params::port` (integer): The BDII service port, default 2170
- `creamce::use_locallogger` (boolean): True if the local logger service must be installed and configured, default is false
- `creamce::info::capability` (list): The list of capability for a CREAM site; it's a list of string, in general with format "name=value", default empty list
- `creamce::vo_software_dir` (string): The base directory for installation of the software used by Virtual Organizations
- `creamce::workarea::shared` (boolean): True if the working area is shared across different Execution Environment instances, typically via an NFS mount; this attribute applies to single-slot jobs, default false
- `creamce::workarea::guaranteed` (boolean): True if the job is guaranteed the full extent of the WorkingAreaTotal; this attribute applies to single-slot jobs, default false
- `creamce::workarea::total` (integer): Total size in GB of the working area available to all single-slot jobs, default 0
- `creamce::workarea::free` (integer): The amount of free space in GB currently available in the working area to all single-slot jobs, default 0 GB
- `creamce::workarea::lifetime` (integer): The minimum guaranteed lifetime in seconds of the files created by single-slot jobs in the working area, default 0 seconds
- `creamce::workarea::mslot_total` (integer): The total size in GB of the working area available to all the multi-slot Grid jobs shared across all the Execution Environments, default 0GB
- `creamce::workarea::mslot_free` (integer): The amount of free space in GB currently available in the working area to all multi-slot jobs shared across all the Execution Environments, default 0 GB
- `creamce::workarea::mslot_lifetime` (integer): The minimum guaranteed lifetime in seconds of the files created by multi-slot jobs in the working area, default 0 seconds

3.4.1 Hardware table

The hardware table contains any information about the resources of the site; the parameter to be used is `creamce::hardware_table`. The hardware table is a hash table with the following structure:

- the key of an entry in the table is the ID assigned to the homogeneous sub-cluster of machines (see GLUE2 execution environment).
- the value of an entry in the table is a hash containing the definitions for the homogeneous sub-cluster, the supported mandatory keys are:
  - `ce_cpu_model` (string): The name of the physical CPU model, as defined by the vendor, for example "XEON"
  - `ce_cpu_speed` (integer): The nominal clock speed of the physical CPU, expressed in MHz
  - `ce_cpu_vendor` (string): The name of the physical CPU vendor, for example "Intel"
  - `ce_cpu_version` (string): The specific version of the Physical CPU model as defined by the vendor
  - `ce_physcpu` (integer): The number of physical CPUs (sockets) in a work node of the sub-cluster
  - `ce_logcpu` (integer): The number of logical CPUs (cores) in a worker node of the sub-cluster
  - `ce_minphysmem` (integer): The total amount of physical RAM in a worker node of the sub-cluster, expressed in MB
  - `ce_os_family` (string): The general family of the Operating System installed in a worker node ("linux", "macosx", "solaris", "windows")
  - `ce_os_name` (string): The specific name Operating System installed in a worker node, for example "RedHat"
  - `ce_os_arch` (string): The platform type of worker node, for example "x86_64"
  - `ce_os_release` (string): The version of the Operating System installed in a worker node, as defined by the vendor, for example "7.0.1406"
  - `nodes` (list): The list of the name of the worker nodes of the sub-cluster

  the supported optional keys are:

  - `ce_minvirtmem` (integer): The total amount of virtual memory (RAM and swap space) in a worker node of the sub-cluster, expressed in MB
  - `ce_outboundip` (boolean): True if a worker node has out-bound connectivity, false otherwise, default true
  - `ce_inboundip` (boolean): True if a worker node has in-bound connectivity, false otherwise default false
  - `ce_runtimeenv` (list): The list of tags associated to the software packages installed in the worker node, the definitions for a tag is listed in the software table, default empty list
  - `ce_benchmarks` (hash): The hash table containing the values of the standard benchmarks ("specfp2000", "specint2000", "hep-spec06"); each key of the table corresponds to the benchmark name, default empty hash
  - `subcluster_tmpdir` (string): The path of a temporary directory shared across worker nodes (see GLUE 1.3)
  - `subcluster_wntmdir` (string): The path of a temporary directory local to each worker node (see GLUE 1.3)

3.4.2 Software table

The software table contains any information about the applications installed in the worker nodes; the parameter to be used is `creamce::software_table`. The software table is a hash with the following structure:

- the key of an entry in the table is the tag assigned to the software installed on the machines (see GLUE2 application environment); tags are used as a reference (`ce_runtimeenv`) in the hardware table.
- the value of an entry in the table is a hash containing the definitions for the software installed on the machines, the supported keys are:
  - `name` (string): The name of the software installed, for example the package name, *mandatory*
  - `version` (string): The version of the software installed, *mandatory*
  - `license` (string): The license of the software installed, default unpublished
  - `description` (string): The description of the software installed, default unpublished

3.4.3 Queues table

The queue table contains definitions for local user groups; the parameter to be declared is `creamce::queues`. The queues table is a hash with the following structure:

- the key of an entry in the table is the name of the batch system queue/partition.
- the value of an entry in the table is a hash table containing the definitions for the related queue/partition, the supported keys for definitions are:
  - `groups` (list): The list of local groups which are allowed to operate the queue/partition, each group *MUST BE* defined in the VO table.

3.4.4 Storage table

The storage table contains any information about the set of storage elements bound to this site. The parameter to be declared is creamce::se_table , the default value is an empty hash. The storage element table is a hash with the following structure: `creamce::queues`. The queues table is a hash with the following structure:

- the key of an entry in the table is the name of the storage element host.
- the value of an entry in the table is a hash table containing the definitions for the related storage element, the supported keys for definitions are:
  - `type` (string): The name of the application which is installed in the storage element ("Storm", "DCache", etc.)
  - `mount_dir` (string): The local path within the Computing Service which makes it possible to access files in the associated Storage Service (this is typically an NFS mount point)
  - `export_dir` (string): The remote path in the Storage Service which is associated to the local path in the Computing Service (this is typically an NFS exported directory).
  - `default` (boolean): True if the current storage element must be considered the primary SE, default false. Just one item in the storage element table can be marked as primary.

3.4.5 Example

```
---
creamce::queues :
    long :  { groups : [ dteam, dteamprod ] }
    short : { groups : [ dteamsgm ] }

creamce::hardware_table :
```

(continues on next page)

(continued from previous page)

```
    subcluster001 : {
        ce_cpu_model : XEON,
        ce_cpu_speed : 2500,
        ce_cpu_vendor : Intel,
        ce_cpu_version : 5.1,
        ce_physcpu : 2,
        ce_logcpu : 2,
        ce_minphysmem : 2048,
        ce_minvirtmem : 4096,
        ce_os_family : "linux",
        ce_os_name : "CentOS",
        ce_os_arch : "x86_64",
        ce_os_release : "7.0.1406",
        ce_outboundip : true,
        ce_inboundip : false,
        ce_runtimeenv : [ "tomcat_6_0", "mysql_5_1" ],
        subcluster_tmpdir : /var/tmp/subcluster001,
        subcluster_wntmdir : /var/glite/subcluster001,
        ce_benchmarks : { specfp2000 : 420, specint2000 : 380, hep-spec06 : 780 },
        nodes : [ "node-01.mydomain", "node-02.mydomain", "node-03.mydomain" ]
        # Experimental support to GPUs
        accelerators : {
            acc_device_001 : {
                type : GPU,
                log_acc : 4,
                phys_acc : 2,
                vendor : NVidia,
                model : "Tesla k80",
                version : 4.0,
                clock_speed : 3000,
                memory : 4000
            }
        }
    }

creamce::software_table :
    tomcat_6_0 : {
        name : "tomcat",
        version : "6.0.24",
        license : "ASL 2.0",
        description : "Tomcat is the servlet container"
    }
    mysql_5_1 : {
        name : "mysql",
        version : "5.1.73",
        license : "GPLv2 with exceptions",
        description : "MySQL is a multi-user, multi-threaded SQL database server"
    }

creamce::vo_software_dir : /afs

creamce::se_table :
    storage.pd.infn.it : { mount_dir : "/data/mount", export_dir : "/storage/export",
                           type : Storm, default : true }
    cloud.pd.infn.it : { mount_dir : "/data/mount", export_dir : "/storage/export",
                         type : Dcache }
```

3.5 CREAM security and accounting

The following parameters configure the security layer and the pool account system:

- `creamce::host_certificate` (string): The complete path of the installed host certificate, default /etc/grid-security/hostcert.pem
- `creamce::host_private_key` (string): The complete path of the installed host key, default /etc/grid-security/hostkey.pem
- `creamce::voms_dir` (string): The location for the deployment of VO description files (LSC), default /etc/grid-security/vomsdir
- `creamce::gridmap_dir` (string): The location for the pool account files, default /etc/grid-security/gridmapdir
- `creamce::gridmap_file` (string): The location of the pool account description file, default /etc/grid-security/grid-mapfile
- `creamce::gridmap_extras` (list): The list of custom entry for the pool account description file, default empty list
- `creamce::gridmap_cron_sched` (string): The cron time parameters for the pool account cleaner, default "5 * * * *"
- `creamce::groupmap_file` (string): The path of the groupmap file, default /etc/grid-security/groupmapfile
- `creamce::crl_update_time` (integer): The CRL refresh time in seconds, default 3600 seconds
- `creamce::ban_list_file` (string): The path of the ban list file, if gJAF/LCMAPS is used, default /etc/lcas/ban_users.db
- `creamce::ban_list` (list): The list of banned users, each item is a Distinguished Name in old openssl format. If not defined the list is not managed by puppet.
- `creamce::use_argus` (boolean): True if Argus authorization framework must be used, false if gJAF must be used, default true
- `creamce::argus::service` (string): The argus PEPd service host name, `mandatory` if `creamce::use_argus` is set to true
- `creamce::argus::port` (integer): The Argus PEPd service port, default 8154
- `creamce::argus::timeout` (integer): The connection timeout in seconds for the connection to the Argus PEPd server, default 30 seconds
- `creamce::argus::resourceid` (string): The ID of the CREAM service to be registered in Argus, default `https://{ce_host}:{ce_port}/cream`
- `creamce::admin::list` (list): The list of service administators Distinguished Name, default empty list
- `creamce::admin::list_file` (string): The path of the file containing the service administrators list, default /etc/grid-security/admin-list
- `creamce::default_pool_size` (integer): The default number of users in a pool account, used if `pool_size` is not define for a VO group, default 100
- `creamce::create_user` (boolean): False if the creation of the users of all the pool accounts is disabled, default True

3.5.1 VO table

The VO table contains any information related to pool accounts, groups and VO data. The parameter to be declared is `creamce::vo_table`, the default value is an empty hash. The VO table is a hash, the key of an entry in the table is the name or ID of the virtual organization, the corresponding value is a hash table containing the definitions for the virtual organization,the supported keys for definitions are:

- `servers` (list): The list of configuration details for the VOMS servers. Each item in the list is a hash; the parameter and any supported keys of a contained hash are `mandatory`. The supported keys are:
  - `server` (string): The VOMS server FQDN
  - `port` (integer): The VOMS server port
  - `dn` (string): The distinguished name of the VOMS server, as declared in the VOMS service certificate
  - `ca_dn` (string): The distinguished name of the issuer of the VOMS service certificate
- `groups` (hash): The list of local groups and associated FQANs, the parameter is `mandatory`, each key of the hash is the group name, each value is a hash with the following keys:
  - `gid` (string): The unix group id, `mandatory`
  - `fqan` (list): The list of VOMS Fully Qualified Attribute Names. The items in the list are used to compose the group map file. The parameter is `mandatory`
- `users` (hash): The description of pool accounts or a static users, the parameter is `mandatory`, each key of the hash is the pool account prefix or the user name for a static user, each value is a hash with the following keys:
  - `name_pattern` (list): The pattern used to create the user name of the pool account, the variables used for the substitutions are `prefix`, the pool account prefix, and `index`, a consecutive index described below; the expression is explained in the ruby guide, default value is

    ```
    %<prefix>s%03<index>d
    ```

  - `primary_fqan` (list): The list of the primary VOMS Fully Qualified Attribute Names associated with the user of the pool account. The attributes are used to calculate the primary group of the user and to compose the grid-map file. The mapping between FQANs and groups refers to the `groups` hash for the given VO. For further details about the mapping algorithm refer to the authorization guide. The parameter is `mandatory`
  - `secondary_fqan` (list): The list of the secondary VOMS Fully Qualified Attribute Names associated with the user of the pool account. The items in the list are used to calculate the secondary groups of the user. The parameter is optional.
  - `create_user` (boolean): False if the creation of the user is disabled for the current pool account, default is True
  - `pub_admin` (boolean): True if the pool account is the defined administrator account, default false, just one administrator account is supported. The first user in the pool account, or the static user if it is the case, is selected for publishing information about VO tags.
  - `accounts` (list): The list of SLURM accounts associated with this set of users, further details can be found in the SLURM specific section

  A pool account can be defined in two different ways. If the user IDs are consecutive the parameters required are:

  - `first_uid` (integer): The initial number for the unix user id of the pool account, the other ids are obtained incrementally with step equals to 1

  - `pool_size` (integer): The number of user in the current pool account, the default value is global definition contained into `creamce::default_pool_size`. If the value for the pool size is equal to zero the current definition must be considered for a static user.

  If the user IDs are not consecutive their values must be specified with the parameter:

  - `uid_list` (list): The list of user ID; the pool account size is equal to the number of elements of the list.

  In any case the user name is created using the pattern specified by the parameter `name_pattern` where the index ranges from 1 to the pool account size (included). It is possible to shift the range of the indexes using the parameter `creamce::username_offset`.

- `vo_app_dir` (_string_): The path of a shared directory available for application data for the current Virtual Organization, as describe by Info.ApplicationDir in GLUE 1.3.
- `vo_default_se` (string): The default Storage Element associated with the current Virtual Organization. It must be one of the key of the storage element table

3.5.2 Example

```
---
creamce::use_argus :                    false
creamce::default_pool_size :            10
creamce::username_offset :              1

creamce::vo_table :
    dteam : {
        vo_app_dir : /afs/dteam,
        vo_default_se : storage.pd.infn.it,
        servers : [
                      {
                          server : voms.hellasgrid.gr,
                          port : 15004,
                          dn : /C=GR/O=HellasGrid/OU=hellasgrid.gr/CN=voms.hellasgrid.
↪gr,
                          ca_dn : "/C=GR/O=HellasGrid/OU=Certification Authorities/
↪CN=HellasGrid CA 2016"
                      },
                      {
                          server : voms2.hellasgrid.gr,
                          port : 15004,
                          dn : /C=GR/O=HellasGrid/OU=hellasgrid.gr/CN=voms2.
↪hellasgrid.gr,
                          ca_dn : "/C=GR/O=HellasGrid/OU=Certification Authorities/
↪CN=HellasGrid CA 2016"
                      }
        ],
        groups : {
            dteam : { fqan : [ "/dteam" ], gid : 9000 },
            dteamsgm : { fqan : [ "/dteam/sgm/ROLE=developer" ], gid : 9001 },
            dteamprod : { fqan : [ "/dteam/prod/ROLE=developer" ], gid : 9002 }
        },
        users : {
            dteamusr : { first_uid : 6000, primary_fqan : [ "/dteam" ],
                         name_pattern : "%<prefix>s%03<index>d" },
            dteamsgmusr : { first_uid : 6100, pool_size : 5,
                            primary_fqan : [ "/dteam/sgm/ROLE=developer" ],
                            secondary_fqan : [ "/dteam" ],
```

(continues on next page)

(continued from previous page)

```
                              name_pattern : "%<prefix>s%02<index>d",
                              pub_admin : true },
            dteamprodusr : { primary_fqan : [ "/dteam/prod/ROLE=developer" ],
                             secondary_fqan : [ "/dteam" ],
                             name_pattern : "%<prefix>s%02<index>d",
                             uid_list  : [ 6200, 6202, 6204, 6206, 6208 ] }
        }
    }
```

3.6 CREAM with TORQUE

The TORQUE cluster must be install before the deployment of CREAM, there's no support in the CREAM CE puppet module for the deployment of TORQUE. Nevertheless the module may be used to configure the TORQUE client on CREAM CE node if and only if the node is different from the TORQUE server node. The YAML parameter which enables the TORQUE client configuration is `torque::config::client`, if it is set to false the configuration is disabled, the default value is true. The CREAM CE puppet module can create queues and pool accounts in TORQUE, the YAML parameter is `torque::config::pool`, if it is set to false the feature is disabled, the default value is true.

Other configuration parameters for TORQUE are:

- `torque::host` (string): The TORQUE server host name, default value is the host name.
- `torque::multiple_staging` (boolean): The BLAH parameter for multiple staging, default false
- `torque::tracejob_logs` (integer): The BLAH parameter for tracejob, default 2
- `munge::key_path` (string): The location of the munge key. If TORQUE client configuration is enabled the path is used to retrieve the manually installed key; `mandatory` if `torque::config::client` is set to true.

3.7 CREAM with SLURM

The SLURM cluster must be install before the deployment of CREAM, there's no support in the CREAM CE puppet module for the deployment of SLURM. The module provides an experimental feature for configuring SLURM users and accounts if the accounting subsystem is enabled in SLURM. The YAML parameter which enables the experimental feature is `slurm::config_accounting`, the default value is false. If it is set to true each user of the pool account is replicated in the SLURM accounting subsystem. The site administrator can associate to the any replicated user one or more SLURM accounts in two different ways:

- Specifying a list of accounts already created in SLURM. The list of SLURM accounts associated to the new user is specified by the parameter `accounts` of the `users` definition of the VO table; the parameter is mandatory in this case.
- Delegating the creation of the SLURM accounts to the puppet module. The module creates a SLURM account for each VO and a SLURM sub-account for each group in a given VO. The parameter to be set for enabling the automatic creation of account is `slurm::standard_accounts`, its default value is false.

3.8 CREAM with HTCondor

The HTCondor cluster must be install before the deployment of CREAM, there's no support in the CREAM CE puppet module for the deployment of HTCondor.

The configuration parameters for HTCondor are:

- `condor::deployment_mode` (string): The queue implementation model used, the value can be "queue_to_schedd" or "queue_to_jobattribute", the default is "queue_to_schedd"
- `condor_queue_attr` (string): The classad attribute used to identify the queue, when the model used is "queue_to_jobattribute", `mandatory` if the deployment mode is "queue_to_jobattribute"
- `condor_user_history` (boolean): True if condor_history should be used to get the final state info about the jobs, the default is false
- `condor::config::dir` (string): The directory containing the configuration files for the HTCondor installation, the default is /etc/condor/config.d
- `condor::command_caching_filter` (string): The executable for caching the batch systems commands, if not specified the caching mechanism is disabled

3.9 Experimental features

The features described in this section are subject to frequent changes and must be considered unstable. Use them at your own risk.

3.9.1 GPU support configuration

The GPU support in the information system (BDII) can be enabled with the configuration parameter `creamce::info::glue21_draft`(boolean), the default value is false. The GPU resources must be described in the hardware table, inserting in the related sub-cluster hashes the following parameter:

- `accelerators` (_hash_): The hash table containing the definitions for any accelerator device mounted in the sub-cluster. Each item in the table is a key-value couple. The key is the accelerator ID of the device and the value consists on a hash table with the following mandatory definitions:
  - `type` (string): The type of the device (GPU, MIC, FPGA)
  - `log_acc` (integer): The number of logical accelerator unit in the sub-cluster
  - `phys_acc` (integer): The number of physical accelerator device (cards) in the subcluster
  - `vendor` (string): The vendor ID
  - `model` (string): The model of the device
  - `version` (string): The version of the device
  - `clock_speed` (integer): The clock speed of the device in MHz
  - `memory` (integer): The amount of memory in the device in MByte

CHAPTER 4

CREAM User's Guide

4.1 CREAM Command Line Interface Guide

This section briefly explains the sequence of operations to be performed by a user to submit and then manage jobs on CREAM based CEs, referring to the C++ Command Line Interface (CLI).

4.1.1 Before starting: get your user proxy

Before using any of the CREAM client commands, it is necessary to have a valid proxy credential available on the client machine. You can create it using the `voms-proxy-init` command. If you already have a valid proxy available on your machine just make the `X509_USER_PROXY` environment variable point to it.

In order to get a proxy certificate issued by VOMS, you should have in the directory `/etc/vomses` the proper VOMS file containing a line as follows:

```
"EGEE" "kuiken.nikhef.nl" "15001" "/O=dutchgrid/O=hosts/OU=nikhef.nl/CN=kuiken.nikhef.
↪nl" "EGEE" "22"
```

or the corresponding line for your VO. You also need to install the VO related `.lsc` files in the `/etc/grid-security/vomsdir/<VO>` directory. In a standard EMI UI installation, these settings should be already there.

Make moreover sure you have in the directory `$HOME/.globus` your certificate/key pair, i.e. the following files:

- usercert.pem
- userkey.pem

Note that file permissions are important: the two files must have respectively 0600 and 0400 permissions.

Then you can issue the VOMS client command (you will be prompted for the pass-phrase):

```
$ voms-proxy-init -voms dteam
Enter GRID pass phrase:
```

(continues on next page)

(continued from previous page)

```
Your identity: /C=IT/O=INFN/OU=Personal Certificate/L=Padova/CN=Massimo Sgaravatto
Creating temporary proxy .................................................................
↪.................... Done
Contacting  voms2.hellasgrid.gr:15004 [/C=GR/O=HellasGrid/OU=hellasgrid.gr/CN=voms2.
↪hellasgrid.gr] "dteam" Done
Creating proxy ............................ Done

Your proxy is valid until Sat Apr 30 05:05:49 2011

$ voms-proxy-info -all
subject   : /C=IT/O=INFN/OU=Personal Certificate/L=Padova/CN=Massimo Sgaravatto/
↪CN=proxy
issuer    : /C=IT/O=INFN/OU=Personal Certificate/L=Padova/CN=Massimo Sgaravatto
identity  : /C=IT/O=INFN/OU=Personal Certificate/L=Padova/CN=Massimo Sgaravatto
type      : proxy
strength  : 1024 bits
path      : /tmp/x509up_u500
timeleft  : 11:59:55
key usage : Digital Signature, Key Encipherment, Data Encipherment
=== VO dteam extension information ===
VO        : dteam
subject   : /C=IT/O=INFN/OU=Personal Certificate/L=Padova/CN=Massimo Sgaravatto
issuer    : /C=GR/O=HellasGrid/OU=hellasgrid.gr/CN=voms2.hellasgrid.gr
attribute : /dteam/Role=NULL/Capability=NULL
attribute : /dteam/italy/Role=NULL/Capability=NULL
attribute : /dteam/italy/INFN-PADOVA/Role=NULL/Capability=NULL
timeleft  : 11:59:55
uri       : voms2.hellasgrid.gr:15004
```

4.1.2 CREAM CLI commands

The most relevant commands to interact with CREAM based CEs are:

-
  ```
  glite-ce-job-submit <jdl_file_1> ... <jdl_file_N>
  ```

  submits N jobs (N must be >=1) to a CREAM based CE. It requires N JDL files as input and returns N CREAM job identifiers.

-
  ```
  glite-ce-delegate-proxy <delegation_Id>
  ```

  allows the user to delegate her proxy credential to the CREAM service. This delegated credential can then be used for job submissions.

-
  ```
  glite-ce-job-status <job_Id_1> ... <job_Id_N>
  ```

  displays information (in particular the states) of N jobs (N must be >=1) previously submitted to CREAM based CEs.

-
  ```
  glite-ce-job-list <host[:port]>
  ```

  lists the identifiers of jobs submitted to a CREAM based CE by the user issuing the command.

-
  ```
  glite-ce-job-cancel <job_Id_1> ... <job_Id_N>
  ```

  cancels N jobs (N must be >=1) previously submitted to CREAM based CEs.

-
  ```
  glite-ce-job-suspend <job_Id_1> ... <job_Id_N>
  ```

  suspends the execution of N jobs (N must be >=1) previously submitted to CREAM based CEs.

-
  ```
  glite-ce-job-resume <job_Id_1> ... <job_Id_N>
  ```

  resumes the execution of N jobs (N must be >=1) which have been previously suspended.

-
  ```
  glite-ce-job-output <job_Id_1> ... <job_Id_N>
  ```

  retrieves the output sandbox files of N jobs (N must be >=1) previously submitted to CREAM based CEs.

-
  ```
  glite-ce-job-purge <job_Id_1> ... <job_Id_N>
  ```

  clears N jobs (N must be >=1) from CREAM based CEs. After this operation the purged jobs canRSQUOt be managed anymore.

-
  ```
  glite-ce-proxy-renew <delegation_Id_1> ... <delegation_Id_N>
  ```

  renews N delegations (N must be >=1), and therefore refreshes the proxy of the jobs submitted to CREAM based CEs using the considered delegations.

-
  ```
  glite-ce-service-info <host[:port]>
  ```

  returns information about the CREAM service (version, status, etc.).

-
  ```
  glite-ce-enable-submission <host[:port]>
  ```

  (re-)enables job submissions on the specified CREAM CE.

-
  ```
  glite-ce-disable-submission <host[:port]>
  ```

  disables job submissions on the specified CREAM CE.

-
  ```
  glite-ce-allowed-submission <host[:port]>
  ```

  checks if jobs submissions on the specified CREAM CE are allowed or have been disabled.

-
  ```
  glite-ce-job-lease <lease_identifier> --endpoint <cream_endpoint> \
                                             --leaseTime <lease_time>
  ```

  create a lease identifier in the CREAM server and associate a time duration to it.

Man pages are available for all the CREAM client commands. You can also access information about the usage of each command by issuing:

```
$ <command> --help
```

4.1.3 Submitting jobs to CREAM based CEs

To submit jobs to CREAM based CEs, the command `glite-ce-job-submit` must be used. The `glite-ce-job-submit` command requires as input one or more job description files; each file describes the job characteristics and requirements through the JDL (Job Description Language). A typical example of a JDL job description file is:

```
[
Type = "Job";
JobType = "Normal";
Executable = "myexe";
StdInput = "myinput.txt";
StdOutput = "message.txt";
StdError = "error.txt";
InputSandbox = {"/users/seredova/example/myinput.txt",
"/users/seredova/example/myexe"};
OutputSandbox = {"message.txt", "error.txt"};
OutputSandboxBaseDestUri = "gsiftp://se.pd.infn.it/data/seredova";
]
```

Such a JDL would make the `myexe` executable be transferred on the remote CREAM CE and be run taking the `myinput.txt` file (also copied from the client node) as input. The standard streams of the job are redirected to files `message.txt` and `error.txt`, and when job completes its execution they are automatically uploaded on `gsiftp://se.pd.infn.it/data/seredova`.

A detailed description of the available JDL attributes and of the rules for building correct JDL files is documented in the CREAM JDL guide.

The jobs submitted to a CREAM based CE are given the delegated credentials of the user who submitted it. These credentials can then be used when operations requiring security support has to be performed by the job.

There are two possible options to deal with proxy delegation:

- asking the automatic delegation of the credentials during the submission operation;
- explicitly delegating credentials, and then asking to rely on these previously delegated credentials on the actual submission operations.

It is highly suggested to rely on this latter mechanism, using the same delegated proxy for multiple job submissions, instead of delegating each time a proxy. Delegating a proxy, in fact, is an operation that can require a non negligible time.

The command `glite-ce-delegate-proxy` is the command to be used to explicitly delegate the user credentials to a CREAM CE.The following shows an example of job submission, performed explicitly delegating credentials. So first of all the credentials are delegated to a CREAM based CE (whose endpoint is specified with the option `--endpoint` (-e):

```
$ glite-ce-delegate-proxy -e cream-ce-01.pd.infn.it mydelid
2006-02-26 15:03:37,286 NOTICE - Proxy with delegation id [mydelid] successfully
delegated to endpoint [https://cream-ce-01.pd.infn.it:8443//ce-cream/services/
↪CREAMDelegation]
```

The identifier of the delegation is then specified with the `--delegationId` (`-D`) option in the job submit operation:

```
$ glite-ce-job-submit -D mydelid -r cream-ce-01.pd.infn.it:8443/cream-lsf-grid02␣
↪myjob1.jdl myjob2.jdl myjob3.jdl
```

The option `-r` (`--resource`) has been used to specify the identifier of the CREAM CE where the job has to be submitted to. `myjob1.jdl myjob2.jdl myjob3.jdl` are the 3 JDL files describing the jobs to be submitted.

The command returns the CREAM job identifiers associated with these jobs (e.g. `https://cream-ce-01.pd.infn.it:8443/CREAM116j9vgnf`) which identify them in clear and unique way all over the Grid system scope.

In addition the user can associate a lease that she/he has previously created with the command `glite-ce-job-lease` by mean of the option `--leaseId <lease_identifier>`:

```
$ glite-ce-job-submit -D mydelid -r cream-ce-01.pd.infn.it:8443/cream-lsf-grid02 --
↪leaseId <my_lease_identifier>
myjob1.jdl myjob2.jdl myjob3.jdl
```

To create a lease in the CREAM service, with a certain duration of time (expressed in seconds), issue the command:

```
$ glite-ce-job-lease --endpoint cream-27.pd.infn.it --leaseTime 3600 myLID
You requested lease time [3600] for lease ID [myLID]
CREAM negotiated the lease time to [3600]
```

The above command has created a lease on `cream-27.pd.infn.it` named “myLID” and lasting 1 hour.

4.1.4 Monitoring jobs

Passing the CREAM job identifiers returned by the `glite-ce-job-submit` command to the `glite-ce-job-status` command, it is possible to monitor the submitted jobs. Several (static and dynamic) information can be shown, depending on the chosen verbosity level. The verbosity level can be 0 (less verbosity), 1 or 2 (most verbosity). Please note that specifying 0 as verbosity level means calling on the CREAM service a faster operation than when using 1 or 2 as verbosity level. The most relevant attribute is the job status.

The following is an example of job status operation, specifying 1 as verbosity level:

```
$ glite-ce-job-status -L 1 https://cream-02.pd.infn.it:8443/CREAM738582717
****** JobID=[https://cream-02.pd.infn.it:8443/CREAM738582717]
Current Status = [DONE-FAILED]
ExitCode = [N/A]
FailureReason = [lsf_reason=256; Cannot move ISB (${globus_transfer_cmd}
gsiftp://cream-02.pd.infn.it//CREAMTests/Exe1/ssh1.sh file:///home/infngrid001/home_
↪cream_738582717/CREAM738582717/ssh1.sh):
error: globus_ftp_client: the server responded with an error 500 500-Command failed.␣
↪: globus_l_gfs_file_open failed.
500-globus_xio: Unable to open file //CREAMTests/Exe1/ssh1.sh
500-globus_xio: System error in open: No such file or directory
500-globus_xio: A system call failed: No such file or directory 500 End.]
Grid JobID = [N/A]

Job status changes:
-------------------
Status = [REGISTERED] - [Tue 22 Jan 2008 15:55:08] (1201013708)
Status = [PENDING] - [Tue 22 Jan 2008 15:55:08] (1201013708)
Status = [IDLE] - [Tue 22 Jan 2008 15:55:11] (1201013711)
Status = [RUNNING] - [Tue 22 Jan 2008 15:55:18] (1201013718)
Status = [DONE-FAILED] - [Tue 22 Jan 2008 16:03:10] (1201014190)

Issued Commands:
-------------------
*** Command Name = [JOB_REGISTER]
Command Category = [JOB_MANAGEMENT]
Command Status = [SUCCESSFULL]
*** Command Name = [JOB_START]
Command Category = [JOB_MANAGEMENT]
Command Status = [SUCCESSFULL]
```

In this example it is interesting to note that the job failed (as reported by the `Current Status` field) for the problem reported in the `FailureReason` field: the file to be transferred was not found.

Instead of explicitly specifying the identifiers of the jobs to monitor, the user can also ask to monitor all her jobs, in case specifying conditions (on the submission date and/or on the job status) that must be met. For example to monitor all jobs, whose status is DONE-OK or DONE-FAILED, submitted to the `grid005.pd.infn.it` CREAM CE between July 23, 2005 10:00 and July 28, 2005 11:00, the following command must be issued:

```
$ glite-ce-job-status --all -e grid005.pd.infn.it:8443 --from '2005-07-23 10:00:00' \
                      --to '2005-07-28 11:00:00' -s DONE-OK:DONE-FAILED
```

4.1.5 Retrieving output of jobs

User can choose to save the output sandbox (OSB) files on a remote server, or save them in the CREAM CE node. In the latter case these files can then be retrieved using the `glite-ce-job-output` command. For example the following command retrieves the output sandbox files of the specified job from the relevant CREAM CE node:

```
$ glite-ce-job-output https://cream-38.pd.infn.it:8443/CREAM295728364
2011-01-29 10:09:50,394 INFO - For JobID [https://cream-38.pd.infn.it:8443/
↪CREAM295728364]
output will be stored in the dir ./cream-38.pd.infn.it_8443_CREAM295728364
```

This command can be used also to retrieve output produced by multiple jobs, by specifying multiple job identifiers as command's arguments

4.1.6 Getting job identifiers

If a user is interested to get the identifiers of all her jobs submitted to a specific CREAM CE, she can use the `glite-ce-job-list` command. For example the following command returns the identifiers of all the jobs submitted to the specified CREAM CE, owned by the user issuing the command:

```
$ glite-ce-job-list grid005.pd.infn.it:8443
```

4.1.7 Cancelling jobs

In some cases it might be needed to cancel jobs which have been previously submitted to CREAM based CEs. This can be achieved via the `glite-ce-job-cancel` command. E.g., the command:

```
$ glite-ce-job-cancel https://grid005.pd.infn.it:8443/CREAM115j5vfnf
```

cancels the specified job.

4.1.8 Suspending and resuming jobs

A running or idle job can be suspended (i.e. its execution will be stopped), and be resumed (i.e. it will run again) later. This can be achieved with the `glite-ce-job-suspend` and `glite-ce-job-resume` commands. The following example shows that after having issued the `glite-ce-job-suspend` command, after a while the job status becomes `HELD`.

```
$ glite-ce-job-suspend https://cream-ce-01.pd.infn.it:8443/CREAM11a79tnb2
Are you sure you want to suspend specified job(s) [y/n]: y
$ glite-ce-job-status -L 0 https://cream-ce-01.pd.infn.it:8443/CREAM11a79tnb2
****** JobID=[https://cream-ce-01.pd.infn.it:8443/CREAM11a79tnb2]
Status = [HELD]
```

Issuing the =glite-ce-job-resume= command, the job will run/will be idle again:

```
$ glite-ce-job-resume https://cream-ce-01.pd.infn.it:8443/CREAM11a79tnb2
Are you sure you want to resume specified job(s) [y/n]: y
$ glite-ce-job-status -L 0 https://cream-ce-01.pd.infn.it:8443/CREAM11a79tnb2
****** JobID=[https://cream-ce-01.pd.infn.it:8443/CREAM11a79tnb2]
Status = [REALLY-RUNNING]
```

4.1.9 Purging jobs

A CREAM job can be monitored (via the =glite-ce-job-status=) even after it has completed its execution. A job gets "lost"; (i.e. it is not possible to monitor or manage it anymore) only when the user who submitted it decides to explicitly clear it, or when the CREAM system administrator decides to do this purging operation. A user can purge her own jobs, using the `glite-ce-job-purge` command. E.g., after having issued the command:

```
$ glite-ce-job-purge https://cream-ce-01.pd.infn.it:8443/CREAM116jbi4o0
```

the specified job canRSQUOt be managed anymore (e.g. it is not possible to check its status anymore).

4.1.10 Renewing proxies

It is possible that long jobs may outlive the validity of the initial delegated credentials; if so the job will die prematurely. To avoid this it is possible to renew the proxy of jobs submitted to CREAM CEs with the `glite-ce-proxy-renew` command. E.g. the following command:

```
$ glite-ce-proxy-renew -e cream-ce-01.pd.infn.it:8443 mydelid
```

renews the proxy of all the jobs having `mydelid` as delegation id. It must be stressed that for jobs submitted to CREAM based CEs via the Workload Management System (WMS), proxy renewal is automatically dealt by the middleware.

4.1.11 Handling job identifiers

Handling the job identifiers directly quickly becomes tedious. To avoid this, you can make the `glite-ce-job-submit` and `glite-ce-job-list` commands append the job Id(s) to a named file using the `--output` (`-o`) option. On the other side, the CREAM client commands which take job identifier(s) as argument accept also the `--input` (`-i`) option which allows the job identifier(s) to be read from a file. The following shows an example:

```
$ glite-ce-job-submit -a -r cream-ce-01.pd.infn.it:8443/cream-lsf-grid02 -o idfile␣
↪myjob.jdl
https://cream-ce-01.pd.infn.it:8443/CREAM116jbs5b9
```

The returned job id got also inserted in the specified file (`idfile`), which can be specified with the `--input` (`-i`) option e.g. with the `glite-ce-job-status` command:

```
$ glite-ce-job-status -i idfile
****** JobID=[https://cream-ce-01.pd.infn.it:8443/CREAM116jbs5b9]
Status=[REALLY-RUNNING]
```

4.1.12 Restricting job submissions

In order to prevent that a CREAM CE gets overloaded, the CREAM CE administrator can set a specific policy to disable new job submissions when certain conditions are met. If submissions are disabled because of that, if newer job submissions are attempted, users will get an error message such as:

```
$ glite-ce-job-submit -a -r cream-38.pd.infn.it:8443/cream-pbs-creamtest1 oo.jdl
MethodName=[jobRegister] ErrorCode=[0] Description=[The CREAM service cannot accept␣
↪jobs at the moment]
FaultCause=[Threshold for Load Average(1 min): 30 => Detected value for Load␣
↪Average(1 min): 31.13]
Timestamp=[Sat 29 Jan 2011 11:55:18]
```

In order to avoid degrading the performance of the system, the specified policy is not evaluated for each job submission, but instead it is evaluated and imposed from time to time (so it might happen that for a short time job submissions are allowed even if the specified threshold has been reached). CREAM "super-users"; can also disable newer job submissions via the command `glite-ce-disable-submission`. Submissions can then be re-enabled by a CREAM "super-user"; via the command `glite-ce-enable-submission`. To check if job submissions on a specific CREAM CE are allowed, the command `glite-ce-allowed-submission` can be used.

```
$ glite-ce-disable-submission grid006.pd.infn.it:8443
Operation for disabling new submissions succeeded

$ glite-ce-allowed-submission grid006.pd.infn.it:8443
Job Submission to this CREAM CE is disabled

$ glite-ce-enable-submission grid006.pd.infn.it:8443
Operation for enabling new submissions succeeded

$ glite-ce-allowed-submission grid006.pd.infn.it:8443
Job Submission to this CREAM CE is enabled
```

It must be stressed that if job submissions to a specific CREAM CE are disabled, all other operations (job status, job cancellations, etc.) can still be performed.

4.1.13 Getting information about the CREAM service

It is possible to get information about the CREAM service (interface and service version, status, etc) using the `glite-ce-service-info` command, e.g.:

```
$ glite-ce-service-info cream-13.pd.infn.it:8443
Interface Version = [2.1]
Service Version = [1.12]
Description = [CREAM 2]
Started at = [Tue Nov 10 14:42:12 2009]
Submission enabled = [YES]
Status = [RUNNING]
Service Property = [SUBMISSION_THRESHOLD_MESSAGE]->
[Threshold for Load Average
(1 min): 10 => Detected value for Load Average(1 min): 0.03
Threshold for Load Average(5 min): 10 => Detected value for Load Average(5 min): 0.03
Threshold for Load Average(15 min): 10 => Detected value for Load Average(15 min): 0.
↪00
Threshold for Memory Usage: 95 => Detected value for Memory Usage: 57.41%
Threshold for Swap Usage: 95 => Detected value for Swap Usage: 2.02%
Threshold for Free FD: 500 => Detected value for Free FD: 204500
```

(continues on next page)

(continued from previous page)

```
Threshold for tomcat FD: 800 => Detected value for Tomcat FD: 107
Threshold for FTP Connection: 30 => Detected value for FTP Connection: 1
Threshold for Number of active jobs: -1 => Detected value for Number of active jobs: 0
Threshold for Number of pending commands: -1 => Detected value for Number of pending␣
↪commands: 0
```

4.1.14 CREAM CLI configuration files

The configuration of the CREAM UI is accomplished via three possible configuration files:

- A general configuration file. This file is looked for in `/etc/glite_cream.conf`
- A VO specific configuration file. This file is looked for in `/etc/<VO>/glite_cream.conf`
- A user specific configuration file. This file is looked for in the following order:
  - The file specified with the `--conf` option of the considered command
  - The file referenced by the `$GLITE_CREAM_CLIENT_CONFIG` environment variable
  - `$HOME/.glite/<VO>/glite_cream.conf` (if the VO is defined), or `$HOME/.glite/glite_cream.conf` otherwise

Each of these files is a classad containing definitions. If the same attribute is defined in more configuration file, the definition in the user specific configuration file (if any) is considered. Likewise the definitions in the VO specific configuration file have higher priority than the ones specified in the general configuration file. It must be noted that one or more (even all) of these three configuration files can be missing.

We list here the possible attributes that can be specified in the configuration files:

Name	Description	Default
CREAM_URL_PREFIX	the prefix to the `<hostname>:<port>` to build the CREAM service endpoint	https://
CREAMDELEGATION_URL_PREFIX	the prefix to the `<hostname>:<port>` to build the CREAM delegation service endpoint	https://
DEFAULT_CREAM_TCPPORT	the port to be appended to the hostname (if not specified by the user) to build the CREAM and CREAM delegation service endpoint	8443
CREAM_URL_POSTFIX	the postfix to be appended to the `<hostname>:<port>` to build the CREAM service endpoint	/ce-cream/services/CREAM2
CREAMDELEGATION_URL_POSTFIX	the postfix to be appended to the `<hostname>:<port>` to build the CREAM delegation service endpoint	/ce-cream/services/gridsite-delegation
JDL_DEFAULT_ATTRIBUTES	the classad that must be included by default in the user's JDLs	empty classad
STATUS_VERBOSITY_LEVEL	the default verbosity level to be used for the `glite-ce-job-status` command	0
UBERFTP_CLIENT	is the pathname of the `uberftp` client executable	/usr/bin/uberftp
SUBMIT_LOG_DIR	the directory where by default the log file `glite-ce-job-submit_CREAM_<username>_<date>_<time>.log` (created when the `--debug` option is used with the `glite-ce-job-submit` command) is created	/tmp/glite_cream_cli_logs
DELEGATE_LOG_DIR	the directory where by default the log file `glite-ce-delegate-proxy_CREAM_<username>_<date>_<time>.log` (created when the `--debug` option is used with the `glite-ce-delegate-proxy` command) is created	/tmp/glite_cream_cli_logs
STATUS_LOG_DIR	the directory where by default the log file `glite-ce-job-status_CREAM_<username>_<date>_<time>.log` (created when the `--debug` option is used with the glite-ce-job-status command) is created	/tmp/glite_cream_cli_logs
LIST_LOG_DIR	the directory where by default the log file `glite-ce-job-list_CREAM_<username>_<date>_<time>.log` (created when the `--debug` option is used with the `glite-ce-job-list` command) is created	/tmp/glite_cream_cli_logs
SUSPEND_LOG_DIR	the directory where by default the log file `glite-ce-job-suspend_CREAM_<username>_<date>_<time>.log` (created when the `--debug` option is used with the `glite-ce-job-suspend` command) is created	/tmp/glite_cream_cli_logs
RESUME_LOG_DIR	the directory where by default the log file `glite-ce-job-resume_CREAM_<username>_<date>_<time>.log` (created when the `--debug` option is used with the `glite-ce-job-resume` command) is created	/tmp/glite_cream_cli_logs
CANCEL_LOG_DIR	the directory where by default the log file `glite-ce-job-cancel_CREAM_<username>_<date>_<time>.log` (created when the `--debug` option is used with the `glite-ce-job-cancel` command) is created	/tmp/glite_cream_cli_logs
JOBOUTPUT_LOG_DIR	the directory where by default the log file `glite-ce-job-output_CREAM_<username>_<date>_<time>.log` (created when the `--debug` option is used with the `glite-ce-job-output` command) is created	/tmp/glite_cream_cli_logs
PURGE_LOG_DIR	the directory where by default the log file `glite-ce-job-purge_CREAM_<username>_<date>_<time>.log` (created when the `--debug` option is used with the `glite-ce-job-purge` command) is created	/tmp/glite_cream_cli_logs
ALLOWEDSUB_LOG_DIR	the directory where by default the log file `glite-ce-allowed-submission_CREAM_<username>_<date>_<time>.log` (created when the `--debug` option is used with the `glite-ce-allowed-submission` command) is created	/tmp/glite_cream_cli_logs
ENABLE_LOG_DIR	the directory where by default the log file `glite-ce-enable-submission_CREAM_<username>_<date>_<time>.`	/tmp/glite_cream_cli_logs

As mentioned above, if the same attribute is defined in more than a configuration file, the definition in the user specific configuration file (if any) has higher priority than the definition in the VO specific configuration file (if any) which has higher priority than the definition in the generic configuration file. If an attribute is not defined anywhere, the default value is considered.

4.2 Use specific functionality of the CREAM CE

4.2.1 Forward of requirements to the batch system

The CREAM CE allows to forward, via tha BLAH component, requirements to the batch system. For this purpose the JDL `CERequirements` attribute, described in the CREAM JDL guide, can be used. For direct submissions to the CREAM CE (e.g. jobs submitted to the CREAM CE using the CREAM CLI `glite-ce-job-submit` command) the CeRequirements attribute is supposed to be filled by the end-user. For jobs submitted to the CREAM CE via the WMS, the `CeRequirements` attribute is instead filled by the WMS, considering the JDL `Requirements` expression and the value of the `CeForwardParameters` attribute in the WMS configuration file.

For example, if in the user JDL there is :

```
Requirements= "other.GlueHostMainMemoryRAMSize > 100 && other.
↪GlueCEImplementationName==\"CREAM\"";
```

and if the WMS configuration file there is:

```
CeForwardParameters  = {"GlueHostMainMemoryVirtualSize","GlueHostMainMemoryRAMSize",
↪"GlueCEPolicyMaxCPUTime"};
```

in the JDL sent by the WMS to CREAM there will be:

```
CeRequirements= "other.GlueHostMainMemoryRAMSize > 100";
```

The `CERequirements` expression received by CREAM is then forwarded to BLAH. Basically BLAH manages the `CERequirements` expression setting some environment variables, which are available and can be properly used by the `/usr/libexec/xxx_local_submit_attributes.sh` script (e.g. `/usr/libexec/pbs_local_submit_attributes.sh` for PBS/Torque, `/usr/libexec/lsf_local_submit_attributes.sh` for LSF). This script must be properly created by the site admin.

For example, considering the following CeRequirements expression:

```
CeRequirements="other.GlueHostMainMemoryRAMSize > 100 && other.GlueCEStateWaitingJobs
↪<10 && \
other.GlueCEImplementationName==\"CREAM\" && other.GlueHostProcessorClockSpeed >=␣
↪2800 && \
(Member(\"FDTD\", other.GlueHostApplicationSoftwareRuntimeEnvironment))";
```

the following settings will be available in `$USR_LOCATION/libexec/xxx_local_submit_attributes.sh`:

```
GlueHostMainMemoryRAMSize_Min='100'
GlueCEStateWaitingJobs_Max='10'
GlueCEImplementationName='CREAM'
GlueHostProcessorClockSpeed_Min='2800'
GlueHostApplicationSoftwareRuntimeEnvironment='"FDTD"'
```

where the value for $USR_LOCATION in a standard installation of a CREAM CE is "/usr". What is printed by the `/usr/libexec/xxx_local_submit_attributes.sh` script is automatically added to the submit command file. For example if the JDL `CeRequirements` expression is:

```
CeRequirements = "(Member(\"FDTD\", other.
↪GlueHostApplicationSoftwareRuntimeEnvironment))";
```

and the =/usr/libexec/pbs_local_submit_attributes.sh= is:

```
#!/bin/sh
if [ "$other.GlueHostApplicationSoftwareRuntimeEnvironment" == "FDTD" ]; then
 echo "#PBS -l software=FDTD"
fi
```

then the PBS submit file that will be used will include:

```
...
...
# PBS directives:
#PBS -S /bin/bash
#PBS -o /dev/null
#PBS -e /dev/null
#PBS -l software=FDTD
....
....
```

where the line:

```
#PBS -l software=FDTD
```

is set via the `/usr/libexec/pbs_local_submit_attributes.sh` script.

Please note that there are no differences if in `CeRequirements` expresssion there is e.g.

```
CeRequirements = other.xyz==\"ABC\"
```

or:

```
CeRequirements = "xyz==\"ABC\"";
```

In both cases in `/usr/libexec/xxx_local_submit_attributes.sh` the variable `xyz` will be set. As shown above, having `x>a` or `x>=a` doesn't make any difference in the setting of the environment variable `x` in the `/usr/libexec/xxx_local_submit_attributes.sh` script. It will be in both cases:

```
x_Min='a'
```

Starting with BLAH v. 1.18 it is possible to forward to the batch system also other attributes not included in the `CeRequiments` JDL attribute. This can be done adding in =/etc/blah.config= the line:

```
blah_pass_all_submit_attributes=yes
```

In this way the `xxx_local_submit_attributes.sh` will see the following environment variables set:

- gridType
- x509UserProxyFQAN
- uniquejobid
- queue
- ceid
- VirtualOrganisation

- ClientJobId
- x509UserProxySubject

It is also possible to specify that only some attributes must be forwarded in the batch system setting in `blah.config` e.g.:

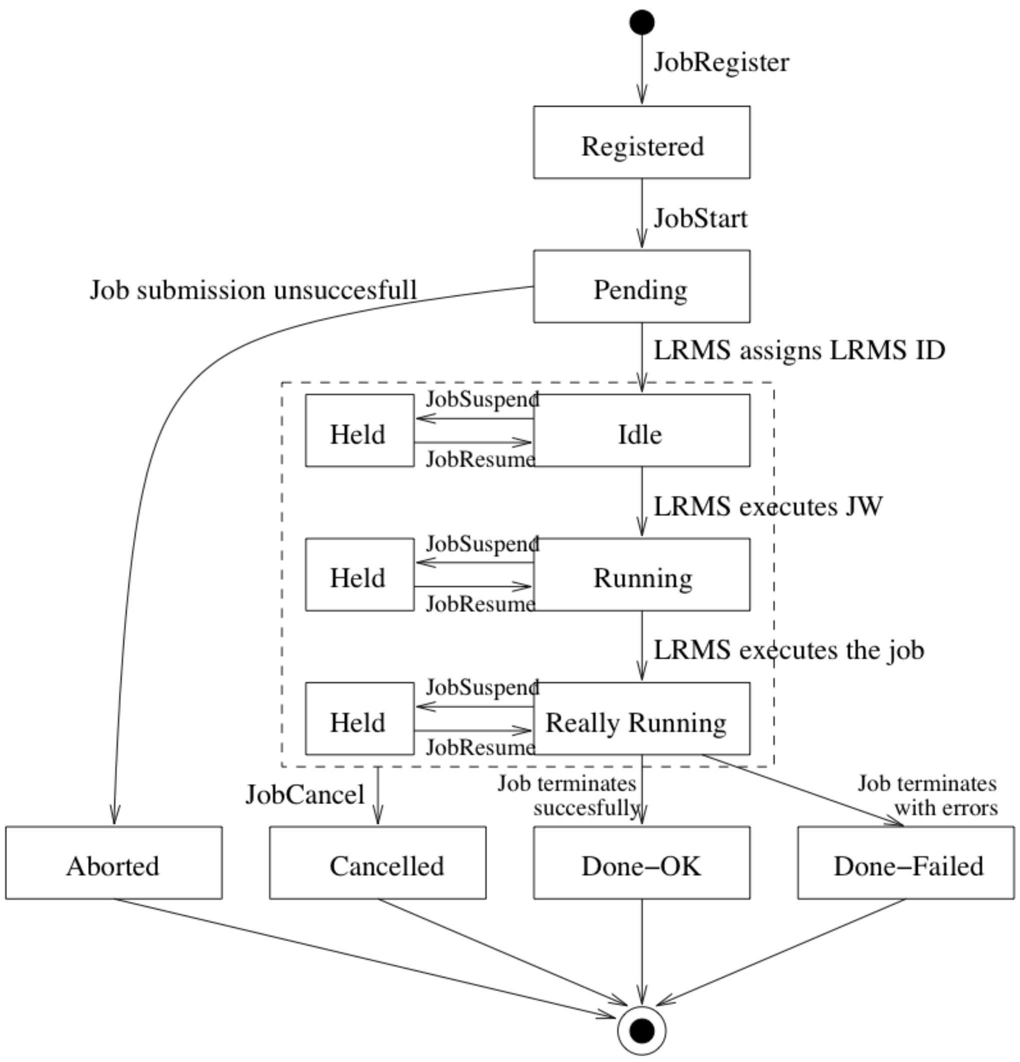
```
blah_pass_submit_attributes[0]="x509UserProxySubject"
blah_pass_submit_attributes[1]="x509UserProxyFQAN"
```

4.3 CREAM job states

Here below is provided a brief description of the meaning of each possible state a CREAM job can enter:

- `REGISTERED`: the job has been registered but it has not been started yet.
- `PENDING` the job has been started, but it has still to be submitted to the LRMS abstraction layer module (i.e. BLAH).
- `IDLE`: the job is idle in the Local Resource Management System (LRMS).
- `RUNNING`: the job wrapper, which "encompasses"; the user job, is running in the LRMS.
- `REALLY-RUNNING`: the actual user job (the one specified as Executable in the job JDL) is running in the LRMS.
- `HELD`: the job is held (suspended) in the LRMS.
- `CANCELLED`: the job has been cancelled.
- `DONE-OK`: the job has successfully been executed.
- `DONE-FAILED`: the job has been executed, but some errors occurred.
- `ABORTED`: errors occurred during the "management"; of the job, e.g. the submission to the LRMS abstraction layer software (BLAH) failed.
- `UNKNOWN`: the job is an unknown status.

The following figure shows the possible job states transitions:


JobRegister
Registered
JobStart
Pending
Job submission unsuccesfull
LRMS assigns LRMS ID
Held
JobSuspend
JobResume
Idle
LRMS executes JW
Held
JobSuspend
JobResume
Running
LRMS executes the job
Held
JobSuspend
JobResume
Really Running
JobCancel
Job terminates succesfully
Job terminates with errors
Aborted
Cancelled
Done−OK
Done−Failed

CHAPTER 5

CREAM JDL Guide

5.1 Introduction

The Job Description Language (JDL) is a high-level, user-oriented language based on Condor classified advertisements (classads) for describing jobs to be submitted to the CREAM CE service. Being the JDL an extensible language the user is allowed to use whatever attribute for the description of a request without incurring in errors from the JDL parser. However, only a certain set of attributes, that we will refer as "supported attributes"; from now on, is taken into account by the CREAM CE service.

Some of the attributes in the JDL are mandatory. If the user does not specify them, CREAM cannot handle the request. For the other attributes the system may find a default value if they are necessary for processing the request.

Before starting with the detailed attribute description please note that a request description is composed by entries that are strings having the format

```
attribute = expression;
```

and are terminated by the semicolon character. The whole description has to be included between square brackets, i.e. `[ <job descr.> ]`. The termination with the semicolon is not mandatory for the last attribute before the closing square bracket].

Attribute expressions can span several lines provided the semicolon is put only at the end of the whole expression. Comments must have a sharp character (#) or a double slash (//) at the beginning of each line. Comments spanning multiple lines can be specified enclosing the text between "/*" and "*/".

Please note that since CREAM exposes a publicly available WSDL interface, no assumption is made in the document (unless explicitly specified) about the client tool used to submit the JDL description of the job.

5.2 Request and Job Types

5.2.1 Type

This a string representing the type of the request described by the JDL, e.g.:

```
Type = "Job";
```

For the time being the only possible value is: `Job` The value for this attribute is case insensitive. If this attribute is not specified in the JDL description, the default value ("Job") will be considered.

5.3 Job Attributes Description

This section reports the detailed description of the JDL attributes that can be specified for describing Job requests. A sub-section for each attribute is provided.

5.3.1 JobType

This a string representing the type of the job described by the JDL, e.g.:

```
JobType = "Normal";
```

At least for the time being the only possible value is: `Normal`. This attribute only makes sense when the Type attribute equals to "Job". The value for this attribute is case insensitive. If not specified in the JDL, it will be set to "Normal"

Mandatory: No

Default: Normal

5.3.2 Executable

This is a string representing the executable/command name. The user can specify an executable that lies already on the remote CE and in this case the absolute path, possibly including environment variables referring to this file should be specified, e.g.:

```
Executable = "usr/local/java/j2sdk1.4.0_01/bin/java";
```

or:

```
Executable = "$JAVA_HOME/bin/java";
```

The other possibility is to provide either an executable located on a remote gridFTP server accessible by the user (HTTPS servers are also supported but this requires to have the GridSite =htcp= client command installed on the WN). In both cases the executable file will be staged from the original location to the Computing Element WN. In both cases only the file name has to be specified as executable. The URI of the executable should be then listed in the InputSandbox attribute expression to make it be transferred. E.g. respectively:

```
Executable = "cms_sim.exe";
InputSandbox = {"file:///home/edguser/sim/cms_sim.exe", ...};
```

Or:

```
Executable = "cms_sim.exe";
InputSandbox = {"gsiftp://neo.datamat.it:5678/tmp/cms_sim.exe", ...};
```

Also descriptions as follows:

```
Executable = "cms_sim.exe";
InputSandbox = {"/home/edguser/sim/cms_sim.exe", ... };
```

are accepted and interpreted as in the first case, i.e. the executable file is available on the local file system.

It is important to remark that if the job needs for the execution some command line arguments, they have to be specified through the =Arguments= attribute. This attribute is mandatory.

Mandatory: Yes

5.3.3 Arguments

This is a string containing all the job command line arguments. E.g. an executable sum that has to be started as:

```
$ sum  N1 N2 -out result.out
```

is described by:

```
Executable = "sum";
Arguments = "N1 N2 -out result.out";
```

If you want to specify a quoted string inside the Arguments then you have to escape quotes with the \ character. E.g. when describing a job like:

```
$ grep -i "my name" *.txt
```

you will have to specify:

```
Executable = "/bin/grep";
Arguments = "-i \"my name\" *.txt";
```

Analogously, if the job takes as argument a string containing a special character (e.g. the job is the tail command issued on a file whose name contains the ampersand character, say file1&file2), since on the shell line you would have to write:

```
$ tail -f file1\&file2
```

in the JDL youRSQUOll have to write:

```
Executable = "/usr/bin/tail";
Arguments = "-f file1\\\&file2";
```

i.e. a \ for each special character. In general, special characters such as:

```
&, |, >, <
```

are only allowed if specified inside a quoted string or preceded by triple . The character ‘ cannot be specified in the JDL.

Some other guidelines for the management of special characters:

- If the arguments string contains one or more spaces, the string must be specified in single quotes
- If the arguments string contains single quotes, then the string must be specified in double quotes (which must be escaped)
- If the arguments string contains spaces and single quotes, then the string must be specified in double quotes (which must be escaped)
- the number of the backquote characters must be an even number

Some examples:

```
Arguments = "\"Problematic character is: '\" " ;

Arguments = "-o '/DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=romanov/CN=427293/
↪CN=Vladimir Romanovskiy' \"It's sunny \"" ;

Arguments = "-o \"'option'\"";
```

Mandatory: No

Default: No

5.3.4 StdInput

This is a string representing the standard input of the job. . This means that the job is run as follows:

```
$ executable < <standard input file>
```

It can be an absolute path possibly including environment variables (wild cards are instead not allowed), i.e. it is already available on the CE, e.g.

```
StdInput = "/var/tpm/jobInput";
```

or just a file name, e.g.

```
StdInput = "myjobInput";
```

and this means that file needs to be made available on the WN where the job is run. Therefore the standard input file has to be added to the InputSandbox file list so that it will be downloaded on the WN. The same rules described for the Executable attribute apply to StdInput.

Mandatory: No

Default: No

5.3.5 StdOutput

This is a string representing the file name where the standard output of the job is saved. The user can specify either a file name or a relative path (with respect to the job working directory on the WN), e.g.:

```
StdOutput = "myjobOutput";

StdOutput = "event1/myjobOutput";
```

Wild cards are not allowed. The value specified for StdError can be the same as the one for StdOutput: this means that the two standard streams of the job are saved in the same file. The user can choose to have this file staged automatically on a GridFTP server specifying a URI for that file in the OutputSandbox attribute expression. E.g.:

```
StdOutput = "myjobOutput";
OutputSandbox = {
"gsiftp://fox.infn.it:5678/home/gftp/myjobOutput",
...
};
```

indicates that `myjobOutput` when the job has completed its execution has to be transferred on `gsiftp://fox.infn.it:5678` in the `/home/gftp` directory.

Mandatory: No

Default: No

5.3.6 StdError

This is a string representing the file name where the standard error of the job is saved. The user can specify either a file name or a relative path (with respect to the job working directory on the WN), e.g.:

```
StdError = "myjobError";

StdError = "event1/myjobError";
```

Wild cards are not allowed. The value specified for StdError can be the same as the one for StdOutput: this means that the two standard streams of the job are saved in the same file. The user can choose to have this file staged automatically on a GridFTP server specifying a URI for that file in the OutputSandboxDestURI attribute expression The same rules as for the StdOutput apply to StdError.

Mandatory: No

Default: No

5.3.7 InputSandbox

This is a string or a list of strings identifying the list of files available on the file system of the client (UI) machine and/or on an accessible gridFTP server (HTTPS servers are also supported but this requires to have the GridSite htcp client command installed on the WN) needed by the job for running. These files hence have to be transferred to the WN before the job is started. Wildcards and environment variables are admitted in the specification of this attribute only if the submission takes place through a client able to resolve them locally before passing the JDL to the CREAM service (e.g. this is the case for the CREAM CLI). Admitted wildcard patterns are the ones supported by the Linux glob function. One can remove the special meaning of the characters: '?', '*', and '[' by preceding them by a backslash.

File names can be provided as URI on a gridFTP/HTTPS server, simple file names, absolute paths and relative paths with respect to the current UI working directory. The InputSandbox file list cannot contain two or more files having the same name (even if in different paths) as when transferred in the job's working directory on the WN they would overwrite each other. This attribute can also be used to accomplish executable and standard input staging to the CE where job execution takes place as explained above. The InputSandbox attribute meaning is strictly coupled with the value of the InputSandboxBaseURI defined in the following that specifies a common location on a gridFTP/HTTPS server where files in the InputSandbox not specified as URI are located.

Support for file transfer from gridftp servers running using user credentials instead of host credentials is also provided1. In this case the distinguish name of such user credentials must be specified in the URI using:

```
?DN=<distinguish name>
```

as shown in the example below.

Here below follows an example of InputSandbox setting:

```
InputSandbox = {
"/tmp/ns.log",
"mytest.exe",
"myscript.sh",
"data/event1.txt",
"gsiftp://neo.datamat.it:5678/home/fpacini/cms_sim.exe ",
"file:///tmp/myconf",
```

(continues on next page)

(continued from previous page)

```
"gsiftp://lxsgaravatto.pd.infn.it:47320/etc/fstab?DN=/C=IT/O=INFN/OU=Personal␣
↪Certificate/L=Padova/CN=Massimo Sgaravatto"
};
InputSandboxBaseURI = "gsiftp://matrix.datamat.it:5432/tmp";
```

It means that:

- `/tmp/ns.log` is located on the UI machine local file system
- `mytest.exe` , `myscript.sh` and `data/event1.txt` are available on `gsiftp://matrix.datamat.it:5432` in the `/tmp` directory
- `/tmp/myconf` is located on the user local file system (explicitly specified using the file:// prefix)
- `/etc/fstab` is available on `gsiftp://lxsgaravatto.pd.infn.it:47320` which is a gridftp server running using user credentials (with the specified distinguish name)

If the InputSandboxBaseURI is not specified than also `mytest.exe`, `myscript.sh` and `data/event1.txt` would be interpreted as located on the user local file system.

Mandatory: No

Default: No

5.3.8 InputSandboxBaseURI

This is a string representing the URI on a gridFTP server (HTTPS servers are also supported but this requires to have the GridSite htcp client command installed on the WN) where the InputSandbox files that have been specified as simple file names and absolute/relative paths are available for being transferred on the WN before the job is started. E.g.

```
InputSandbox = {
 ...
 "data/event1.txt",
 ...
 };
InputSandboxBaseURI = "gsiftp://matrix.datamat.it:5432/tmp";
```

makes CREAM consider

```
"gsiftp://matrix.datamat.it:5432/tmp/data/event1.txt"
```

for the transfer on the WN.

Support for file transfer from gridftp servers running using user credentials instead of host credentials is also provided1. In this case the distinguish name of such user credentials must be specified in the URI using:

```
?DN=<distinguish name>
```

as shown in the example below. E.g.

```
InputSandbox = {
 ...
 "data/event2.txt",
 ...
 };
InputSandboxBaseURI  = "gsiftp://lxsgaravatto.pd.infn.it:47320/tmp?DN=/C=IT/O=INFN/
↪OU=Personal Certificate/L=Padova/CN=Massimo Sgaravatto";
```

makes CREAM consider

```
"gsiftp://lxsgaravatto.pd.infn.it:47320/tmp/data/event2.txt"
```

for the transfer on the WN, where that gridftp server has been started using user credentials (with the specified distinguish name)

Mandatory: No

Default: No

5.3.9 OutputSandbox

This is a string or a list of strings identifying the list of files generated by the job on the WN at runtime, which the user wants to save. This attribute can be combined with the OutputSandboxDestURI or the OutputSandboxBaseDestURI to have, upon job completion, the output directly copied to specified locations running a gridFTP server (HTTPS servers are also supported but this requires to have the GridSite htcp client command installed on the WN). Wildcards are admitted in the specification of this attribute only if the OutputSandboxBaseDestURI attribute is used along with the OutputSandbox attribute. Admitted wildcard patterns are the ones supported by the Linux glob function. One can remove the special meaning of the characters: '? ', '*' and '[' by preceding them by a backslash.

File names can be provided as simple file names or relative paths with respect to the current working directory on the executing WN. The OutputSandbox file list should not contain two or more files having the same name (even if in different paths).

Mandatory: No

Default: No

5.3.10 OutputSandboxDestURI

This is a string or a list of strings representing the URI(s) on a gridFTP/HTTPS server where the files listed in the OutputSandbox attribute have to be transferred at job completion. The OutputSandboxDestURI list contains for each of the files specified in the OutputSandbox list the URI (including the file name) where it has to be transferred at job completion. Support for file transfer to gridftp servers running using user credentials instead of host credentials is also provided1. In this case the distinguish name of such user credentials must be specified in the URI using:

```
?DN=<distinguish name>
```

as shown in the example below. E.g.

```
OutputSandbox = {
"myjobOutput",
"run1/event1",
"run2/event2",
};

OutputSandboxDestURI = {
"gsiftp://matrix.datamat.it:5432/tmp/myjobOutput ",
"gsiftp://grid003.ct.infn.it:6789/home/cms/event1",
"gsiftp://lxsgaravatto.pd.infn.it:47320/tmp/event2?DN=/C=IT/O=INFN/
OU=Personal Certificate/L=Padova/CN=Massimo Sgaravatto"
};
```

makes CREAM transfer respectively:

- `myjobOutput` on `matrix.datamat.it` in the directory `/tmp`

- event1 on grid003.ct.infn.it in the directory /home/cms
- event2 on lxsgaravatto.pd.infn.it (gridftp server running using user credentials, with the specified distinguish name) in the directory /tmp

Specifying the URI gsiftp://localhost, the OutputSandboxFile is saved in the gridftp server of the CREAM CE, as shown in the following example:

```
OutputSandbox = {
"file1",
"file2",
};
OutputSandboxDestURI = {
"gsiftp://localhost/file1",
"gsiftp://grid003.ct.infn.it:6789/home/cms/file2"
};
```

In the above example file1 is saved on the gridftp server of the CREAM CE, while file2 is saved on grid003.ct.infn.it (directory /home/cms). The OutputSandboxDestURI list must have the same cardinality as the OutputSandbox list, otherwise the JDL will be considered as invalid. Note that the file name specified in the OutputSandbox can be different from the corresponding destination file name specified in the OutputSandboxBaseDestURI. The OutputSandboxDestURI attribute and the OutputSandboxBaseDestURI cannot be specified together in the same JDL. One (and only one) among the OutputSandboxDestURI and OutputSandboxBaseDestURI attributes must be specified if the OutputSandbox attribute has been specified

Mandatory: No

Default: No

5.3.11 OutputSandboxBaseDestURI

The OutputSandboxBaseDestURI attribute is a string representing the base URI on a gridFTP server, i.e. a directory on the server, where the files listed in the OutputSandbox attribute have to be transferred at job completion. HTTPS servers are also supported but this requires to have the GridSite htcp client command installed on the WN. All the OutputSandbox files are transferred to the location specified by the URI with the same names (only names in a flat directory) as the ones specified in the OutputSandbox. E.g.:

```
OutputSandbox = {
"myjobOutput",
"run1/event1",
};
OutputSandboxBaseDestURI = "gsiftp://matrix.datamat.it:5432/tmp";
```

makes CREAM transfer both files in the /tmp directory of the gridFTP server matrix.datamat.it (note that event1 will go in /tmp and not in /tmp/run1).

Support for file transfer to gridftp servers running using user credentials instead of host credentials is also provided1. In this case the distinguish name of such user credentials must be specified in the URI using:

```
?DN=<distinguish name>
```

as shown in the example below. E.g.

```
OutputSandbox = {
"myjobOutput",
"run1/event1",
```

(continues on next page)

(continued from previous page)

```
};
OutputSandboxBaseDestURI = "gsiftp://lxsgaravatto.pd.infn.it:47320/tmp?DN=/C=IT/
↪O=INFN/OU=Personal Certificate/L=Padova/CN=Massimo Sgaravatto";
```

makes CREAM transfer both files in the `/tmp` directory of the gridFTP server `lxsgaravatto.pd.infn.it` (running using user credentials, with the specified distinguish name).

Specifying the URI =gsiftp://localhost=, the OutputSandboxFile is saved in the gridftp server of the CREAM CE, as shown in the following example:

```
OutputSandbox = {
"file1",
"file2",
};
OutputSandboxBaseDestURI = "gsiftp://localhost";
```

In the above example `file1` and `file2` are saved on the gridftp server of the CREAM CE. The OutputSandboxBaseDestURI attribute and the OutputSandboxDestURI cannot be specified together in the same JDL. One (and only one) among the OutputSandboxDestURI and OutputSandboxBaseDestURI attributes must be specified if the OutputSandbox attribute has been specified.

Mandatory: No

Default: No

5.3.12 Prologue

The Prologue attribute is a string representing the executable/script name of the prologue. The prologue is an executable run within the CREAM job wrapper before the user job is started. It can be used for purposes ranging from application-specific checks that the job environment has been correctly set on the WN to actions like data transfers, database updates or MPI pre script. If the prologue fails the job wrapper terminates. The rules for specification of the Prologue attributes and its relationship with the InputSandbox attribute are exactly the same already described for the Executable attribute.

Mandatory: No

Default: No

5.3.13 PrologueArguments

The PrologueArguments attribute is a string containing all the prologue executable command line arguments. All the rules reported in the description of the Arguments attribute also apply to the PrologueArguments attribute.

Mandatory: No

Default: No

5.3.14 Epilogue

The Epilogue attribute is a string representing the executable/script name of the epilogue. The epilogue is an executable/script run within the CREAM job wrapper after the user job completion. It can be used for purposes ranging from application-specific checks that the job performed correctly to actions like data transfers, database updates or MPI post script. The rules for specification of the Epilogue attributes and its relationship with the InputSandbox attribute are exactly the same already described for the Executable attribute.

Mandatory: No

Default: No

5.3.15 EpilogueArguments

The EpilogueArguments attribute is a string containing all the epilogue executable command line arguments. All the rules reported in the description of the Arguments attribute also apply to the EpilogueArguments attribute.

Mandatory: No

Default: No

5.3.16 Environment

This is a list of string representing environment settings that have to be performed on the execution machine and are needed by the job to run properly. The JobWrapper on the Worker Node performs these settings just before the job is started. Each item of the list is an equality `'VAR_NAME=VAR_VALUE'`. E.g.:

```
Environment  = {
  "JOB_LOG_FILE=/tmp/myjob.log",
  "ORACLE_SID=edg_rdbms_1",
  "JAVABIN=/usr/local/java"
};
```

Mandatory: No

Default: No

5.3.17 PerusalFileEnable

The PerusalFileEnable attribute is a Boolean attribute that allows enabling the job file perusal support in CREAM. File perusal can be used when jobs are submitted to CREAM by the WMS, but it is possible to use this functionality also for jobs submitted directly to CREAM. When this attribute is set to true, i.e.

```
PerusalFileEnable = true;
```

the user can inspect, while the job is running, the files generated by the job on the WN. This is achieved by uploading on a location specified by the attribute PerusalFilesDestURI, at regular time intervals, chunks of the files (specified by the attribute PerusalListFileURI) generated by the job on the WN. The PerusalFileEnable attribute is not mandatory. If not specified in the JDL it is assumed to be set to false.

Mandatory: No

Default: False

5.3.18 PerusalTimeInterval

The PerusalTimeInterval attribute is a positive integer representing the difference in seconds between two subsequent saving (and upload on the location specified by the attribute PerusalFilesDestURI) of the job files generated by the job on the WN. Specifying e.g.

```
PerusalTimeInterval = 10;
```

makes the CREAM JobWrapper save the job files specified through the attribute PerusalListFileURI each 10 seconds and upload them on location specified by the attribute PerusalFilesDestURI, so that they can be inspected by the user.

Mandatory: No (unless PerusalFileEnable is true)

Default: No

5.3.19 PerusalFilesDestURI

The PerusalFilesDestURI attribute is a string representing the URI of the location on a gridFTP or HTTPS server (HTTPS servers are also supported but this requires to have the GridSite htcp command installed on the WN) where the chunks of files generated by the running job on the WN and specified through the attribute PerusalListFileURI have to be copied. E.g.

```
PerusalFilesDestURI = gsiftp://ghemon.cnaf.infn.it/home/glite/peek
```

Mandatory: No (unless PerusalFileEnable is true)

Default: No

5.3.20 PerusalListFileURI

The PerusalListFileURI attribute is a string representing the URI of the file on a gridFTP server, containing the list of files (one for each line), that must be saved and uploaded on the location specified by the attribute PerusalFilesDestURI at regular time intervals. E.g.

```
PerusalListFileURI = "gsiftp://ghemon.cnaf.infn.it/peek/files2peek";
```

Mandatory: No (unless PerusalFileEnable is true)

Default: No

5.3.21 BatchSystem

This is a string representing the Local Resource Management System (LRMS), that is the batch system type (e.g. LSF, PBS, etc.) of the target CREAM CE. Here below follows an example for this attribute:

```
BatchSystem = "pbs";
```

This attributes is mandatory but can be omitted if the CREAM CLI is used, since it retrieves this attribute from the CEID and automatically fills the JDL.

Mandatory: Yes

Default: No

5.3.22 QueueName

This is a string representing the queue name in the Local Resource Management System (LRMS) where the job has to be submitted on the target CREAM CE. Here below follows an example for this attribute:

```
QueueName = "long";
```

. This attributes is mandatory but can be omitted if the CREAM CLI is used, since it retrieves this attribute from the CEID and automatically fills the JDL.

Mandatory: Yes

Default: No

5.3.23 CPUNumber

The CpuNumber attribute is an integer greater than 0 specifying the number of CPUs needed. This attribute can be used in particular for MPI jobs. Please note that this attributes allows allocating the specified number of CPUs. Then it is up to the job using them to run the job (e.g. via mpistart). An example of the JDL setting is:

```
CpuNumber = 5;
```

Mandatory: No

Default: 1

5.3.24 SMPGranularity

The SMPGranularity attribute is an integer greater than 0 specifying the number of cores any host involved in the allocation has to dedicate to the considered job. This attribute canRSQUOt be specified along with the Hostnumber attribute when WholeNodes is false.

Mandatory: No

Default: No

5.3.25 GPUNumber

The GPUNumber attribute is an integer greater than 0 specifying the number of accelerator devices (CUDA) required for executing the current job.

Mandatory: No

Default: No

5.3.26 GPUModel

The GPUModel attribute is a string containing the model of the accelerator device (CUDA) as declared by the vendor.

Mandatory: No

Default: No

5.3.27 WholeNodes

The WholeNodes attribute is a boolean that indicates whether whole nodes should be used exclusively or not.

Mandatory: No

Default: False

5.3.28 HostNumber

HostNumber is an integer indicating the number of nodes the user wishes to obtain for his job. This attribute can't be specified along with the SMPGranularity attribute when WholeNodes is false. Please note that `HostNumber` shouldn't be greater than `CpuNumber`

Mandatory: No

Default: No

5.3.29 CERequirements

The CERequirements attribute is a Boolean ClassAd expression that uses C-like operators. It represents job requirements on resources. Properly configuring the BLAH software, it is possible to instruct the Local Resource Management System to select the most appropriate Worker Node to run the considered job. The CERequirements expression can contain attributes that describe the CE which are prefixed with `other.`. This is an example of CERequirements expression:

```
CERequirements = "other.GlueCEPolicyMaxCPUTime >= 100 && other.
↪GlueHostMainMemoryRAMSize > 2000";
```

The CERequirements attribute can be used when jobs are submitted directly to a CREAM based CE, while for jobs submitted to a CREAM based CE via the WMS, the CERequirements expression is automatically filled by the WMS considering the user's `Requirements` expression, and the value of the `CeForwardParameters` attribute in the WMS configuration file.

For further details refer to the User guide.

Mandatory: No

Default: No

5.3.30 MWVersion

The MWversion attribute is a string whose value is given to the environment value `EDG_MW_VERSION`, defined for the job immediately when it arrives on the WN (i.e. it is not defined in the job wrapper). There can be hooks on the WN which detect this (before any `/etc/profile.d` scripts are run) and set up the appropriate environment for a job.

Mandatory: No

Default: No

5.3.31 OutputData

This attribute allows the user to ask for the automatic upload and registration to the Replica Catalog of datasets produced by the job on the WN. Through this attribute it is possible to indicate for each output file the LFN (Logical File Name) to be used for registration and the SE (Storage Element) on which the file has to be uploaded. The OutputData attribute is not mandatory.

OutputData is a list of classads where each classad contains the following three attributes:

- `OutputFile`
- `StorageElement`
- `LogicalFileName`

These three attributes are only admitted if members of one of the classads composing OutputData. They cannot be specified independently in the job JDL. Here below follows an example of the !OutputData attribute:

```
OutputData = {
[
            OutputFile = "dataset_1.out ";
            LogicalFileName = "lfn:/test/result1";
 ],
[
            OutputFile = "dataset_2.out ";
            StorageElement = "se001.cnaf.infn.it";
 ],
]
            OutputFile = "cms/dataset_3.out";
            StorageElement = "se012.to.infn.it";
            LogicalFileName = "lfn:/cms/outfile1";
 ],
[
            OutputFile = "dataset_4.out ";
 ]
        };
```

If the attribute OutputData is found in the JDL then the JobWrapper at the end of the job calls the Data Management service that copies the file from the WN onto the specified SE and registers it with the given LFN. If the specified LFN is already in use, the DM service registers the file with a newly generated identifier GUID (Grid Unique Identifier). During this process the JobWrapper creates a file (named `DSUpload_<jobid>.out`) with the results of the operation. In case of submission to CREAM through the WMS this file is put automatically in the OutputSandbox attribute list by the UI (and can then be retrieved by the user with the `glite-wms-job-output` command). If instead the job was submitted directly to the CE, the file is put in the OSB directory of the CE node (and therefore can then be retrieved by the user with the `glite-ce-job-output` command) unless the `OutputSandboxBaseDestURI` attribute has been used (in this latter case the specified location is used to store the DSUpload_<jobid>.out file).

Mandatory: No

Default: No

5.3.32 OutputFile

This is a string attribute representing the name of the output file, generated by the job on the WN, which has to be utomatically uploaded and registered by the WMS. Wildcards are not admitted in the specification of this attribute. File names can be provided as simple file names, absolute paths or relative paths with respect to the current working directory.

Mandatory: Yes (only if OutputData has been specified)

Default: No

5.3.33 StorageElement

This is a string representing the URI of the Storage Element where the output file specified in the corresponding OutputFile attribute has to be uploaded by the WMS.

Mandatory: No

Default: No

5.3.34 LogicalFileName

This is a string representing the logical file name (LFN) the user wants to associate to the output file when registering it to the Replica Catalogue. The specified name has to be prefixed by "lfn:" (lowercase). If this attribute is not specified then the corresponding output file is registered with a GUID that is assigned automatically by the Data Management services.

Mandatory: No

Default: No (If not specified a GUID is assigned by DM services)

5.4 JDL Examples

Simple examples of JDL describing different types of jobs and requests are reported in this section.

5.4.1 Example 1

```
[
    Type = "job";
    JobType = "normal";
    Executable = "/sw/command";
    Arguments = "60";
    StdOutput = "sim.out";
    StdError = "sim.err";
    OutputSandbox = { "sim.err", "sim.out" };
    OutputSandboxBaseDestURI = "gsiftp://se1.pd.infn.it:5432/tmp";
    InputSandbox = {
        "file:///home/user/file1",
        "gsiftp:///se1.pd.infn.it:1234/data/file2",
        "/home/user/file3", "file4"
    };
    InputSandboxBaseURI = "gsiftp://se2.cern.ch:5678/tmp";
]
```

With this JDL a "normal" (batch) job will be submitted. Besides the specification of the executable (already available in the file system of the executing node, since not listed in the InputSandbox), and of the standard output/error files, it is specified that the files `file1`, `file2`, `file3`, `file4` will have to be staged on the executing node:

- `file1` and `file3` will be copied from the client (UI) file system
- `file2` will be copied from the specified GridFTP server (`gsiftp:///se1.pd.infn.it:1234/data/file2`)
- `file4` will be copied from the GridFTP server specified as InputSandboxBaseURI (`gsiftp://se2.cern.ch:5678/tmp`)

It is also specified that the file `sim.err` and `sim.out` (specified as OutputSandbox) must be automatically uploaded into `gsiftp://se1.pd.infn.it:5432/tmp` when job completes its execution.

5.4.2 Example 2

```
[
    Type = "job";
    JobType = "normal";
```

(continues on next page)

(continued from previous page)

```
    Executable = "script.sh";
    Arguments = "60";
        StdOutput = "sim.out";
        StdInput = "sim.inp";
        StdError = "sim.err";
    OutputSandbox = {
        "sim.err",
        "sim.out"
    };
    OutputSandboxDestURI = {
        "gsiftp://matrix.datamat.it:5432/tmp/sim.err",
        "gsiftp://grid003.ct.infn.it:6789/home/cms/sim.out",
    };
    InputSandbox = {
        "file:///home/user/file1",
        "gsiftp:///se1.pd.infn.it:1234/data/file2",
        "/home/user/file3",
        "file4",
        "script.sh",
        "sim.inp"
    };
]
```

This JDL is very similar to the previous one. The only differences are the following:

- The executable and the standard input files have been included in the InputSandbox, and therefore they will be staged in the executing node
- Instead of specifying the URL to be used for all the files of the OutputSandbox, it is specified (via the attribute OutputSandboxDestURI) an URI for each file that have to be uploaded (the files listed as OutputSandbox)
- The attribute InputSandBoxBaseURI hasnRSQUOt been specified, so the files `file4`, `script.sh` and `sim.inp` will be copied from the file system of the client (UI) machine.

CHAPTER 6

BLAH User's Guide

6.1 BLAH Introduction

BLAHPD is a light component accepting commands according to the BLAH (Batch Local Ascii Helper) protocol to manage jobs on different Local Resources Management Systems (LRMS). The BLAH service provides a minimal, pragmatically designed common interface for job submission and control to a set of batch systems. Interaction with BLAH is achieved via text commands whose syntax and semantics is described below. Interaction with each one of the various supported batch system is achieved via customized scripts (the existing scripts are all Bourne shell scripts) that take care of executing the following functions:

- job submit
- job hold
- job resume
- job status
- job cancel

6.2 BLAH Portability

The purpose of this writeup is to:

- describe the assumptions that are currently made on the minimal functionality that batch systems must provide;
- describe the details of the command line arguments that are passed to each one of the five scripts that implement the fuctionality described above for a specific batch system;
- provide guidelines for porting of the existing scripts, especially the submit script for which helper shell functions were defined.

As the universe of different batch systems that need to be supported is very small, the most efficient way to get help beyond the information collected in these notes is to interact with the BLAH developers directly via the e-mail address blah-help@mi.infn.it The string XXX, in the following text, is meant to be replaced with the name of the specific batch system being interfaced (pbs, lsf, condor, etc.).

BLAH assumes that batch systems are capable to

- Identify jobs for subsequent operations via a unique, constant identifier that is returned at the time of job submission.
- Transfer and optionally rename a configurable set of files from a given location on the submit node to the initial working directory on the selected worker node before the job execution starts.
- Transfer and optionally rename a configurable set of files from the initial working directory of job on the worker node that is selected for execution to a given location on the submit node.
- Provide status of currently running jobs via an appropriate command.
- Provide a historical track of jobs that were accepted and ran in the past via one or more log files.

BLAH can use the optional batch system functionality of holding (suspend) and resuming jobs via appropriate commands.

BLAH doesn't require batch systems to be able to renew X509 proxies and/or transfer files to and from the worker node *while the job is being executed*. Proxy renewal is taken care of by a proxy renewal daemon that runs at the side of the running job and receives delegations of refreshed proxies during the lifetime of the job.

6.3 The supported batch systems

At the time of writing BLAH supports the following batch systems: LSF, PBS/Torque, SGE, Condor and just recently SLURM. In order to enable BLAH to interact with the selected batch system, it must be configured properly by setting few parameters on its configuration file which is located by default at `/etc/blah.config`. The file contains a list of parameters and their own default value is well defined for a standard setup of BLAH. Their customization is even possible if needed. For example to enable the support of SLURM is need just a specific parameter `slurm_binpath` which informs BLAH about the location where the SLURM executables are located (i.e. scontrol and sacct).

6.3.1 XXX_submit.sh script

Submit a new job request to the batch system.

NOTE: most of the functionality for parsing and handling the submit script arguments is provided by a set of shell functions that are described further below (see SUBMIT SCRIPT HELPER FUNCTIONS). The argument description is provided for reference only. *SYNOPSIS:*

```
XXX_submit.sh -c <command> -q <queue>[-i <stdin_file>]
    [-o <stdout_file>] [-e <stderr_file>] [-x <x509userproxy>]
    [-v <environment> | -V <environment>] [-s <YES | no>]
    [-d <YES | no>] [-w <workdir>] [-n <number_of_MPI_nodes>]
    [-r <YES | no>] [-p <proxyrenew_poll_interval>]
    [-l <minimum_left_lifetime_proxy>] [-j <jobid_in_caller>]
    [-T <location_temp_of_dir>] [-I <list_input_of_additional_files>]
    [-O <list_of_additional_files_output>]
    [-R <list_of_remap_file_output_rules>] [-C <ce_file_requirements>]
    [-- command_arguments]
```

Any argument after '–' will be passed to the user job ("command").

The command line switches have the following meaning (switches listed in alphabetical order):

- `[-C <ce_file_requirements>]`: When this argument is present, the local script XXX_local_submit_attributes.sh is called, and its output is pasted to the batch system submit file. The XXX_local_submit_attributes.sh script is called after sourcing the contents of the 'CE requirements file' first. This file is composed by the BLAH main daemon and sets shell variables that specify attribute bounds derived from the CERequirements attribute in the BLAH submit command. Example format for the 'CE requirements file':

```
GlueHostMainMemoryRAMSize _Min=1000
GlueCEPolicyMaxCPUTime _Max=30
```

- `[-c command]`: (shell) command to be executed as a batch job.
- `[-d <YES | no>]`: Debug option. If set to 'yes', it causes the submit file to be sent to /dev/tty, and no actual submission to the batch system.
- `[-e <stderr_file>]`: Location of the local file that should receive the job STDERR upon job termination.
- `[-I <list_input_of_additional_files>]`: points to an optional, temporary file containing a list of additional files (one per line) that need to be transferred from the submission node to the initial working directory on the worker node. This temp file is removed by the submit script.
- `[-i <stdin_file>]`: Location of a local file to be transferred to the worker node and connected to the STDIN of the job.
- `[-j <jobid_in_caller>]`: String for unique identification of the job by the calling layer. As this string is unique, it is used to name the submit script, that will be created as 'cream_jobID'.
- `[-l <minimum_left_lifetime_proxy>]`: Minimum remaining lifetime of the user proxy (expressed in seconds) before the proxy renewal damon will kill the user job. Defaults to 180 seconds, or 3 minutes.
- `[-n <number_of_MPI_nodes>]`: Number of MPI nodes to be reserved for the job, in case this feature is supported by the underlying batch system.
- `[-O <list_of_additional_files_output>]`: points to an optional, temporary file containing a list of additional files (one per line) that need to be transferred from the working directory on the worker node back to the submit node. This temp file is removed by the submit script. The -R arguments can be used to establish file name remapping.
- `[-o <stdout file>]`: Location of the local file that should receive the job STDOUT upon job termination.
- `[-p <proxyrenew_poll_interval>]`: time interval (expressed in seconds) between attempts of the proxy renewal daemon to check that the use process is still alive. Defaults to 60 seconds.
- `[-q <queue_name>]`: Batch system queue to run the job in.
- `[-R <list_of_remap_file_output_rules>]`: Points to a file containing a list of names is transferred from the worker node to the submit (local) node. The list file will be deleted by the submit script.
- `[-r <YES | no>]`: Disables the entire BLAH proxy renewal machinery when set to 'no'. Defaults to 'yes' if missing.
- `[-T <location_temp_of_dir>]`: sets the directory location for storing temporary files.
- `[-v <environment> | -V <environment>]`: Environment variables to be set for the user job. Two formats are possible:
  - semicolon-separated assignments -v "ENV1=val1;ENV2=val2;..."
  - space-separated assignments -V "ENV1=val1 ENV2=val2 ..."
- `[-x <x509userproxy>]`: Location of the initial X509 user proxy to associate with the job. This file is initially transferred with the job, then (optionally, see -r) renewed via proxy delegation.
- `[-w <workdir>]`: Directory location pre-pended to relative file paths and used as CWD by the submit script.

RETURN VALUE: Termination code is zero on success, nonzero on error. Job identifier, with the identifier string 'BLAHP_JOBID_PREFIX' is returned on STDOUT on success. The job identifier must start with the batch system name followed by a slash (XXX/id).

SUBMIT SCRIPT HELPER FUNCTIONS: A set of shell functions was written to ease the parsing and handling of the submit script options. They allow to write a submit script along the following template:

```
#!/bin/bash
# 1. Source definition of helper functions.
. `dirname $0`/blah_common_submit_functions.sh

# 2. Parse the submit options, that are used to set
#    a list of bls_opt_XYZ shell variables.
bls_parse_submit_options $@

# 3. Set up temporary files. Set a variable with the name of
#    an environment variable that holds the batch system job ID
#    at runtime.
bls_setup_all_files
bls_job_id_for_renewal=XXX_JOBID

# 4. Start writing the submit script to $bls_tmp_file

cat > $bls_tmp_file << end_of_preamble
#!/bin/bash
# PBS job wrapper generated by `basename $0`
# on `/bin/date`
# proxy_string = $bls_opt_proxy_string
# proxy_local_file = $bls_proxy_local_file
# Etc. Etc. Etc.
end_of_preamble

# 5. Handle script local customisations according to -C option
#    as appropriate for the batch system at hand.
if [ ! -z $bls_opt_req_file ] ; then
  echo \#\!/bin/sh >> ${bls_opt_req_file}-temp_req_script
  cat $bls_opt_req_file >> ${bls_opt_req_file}-temp_req_script
  echo "source ${GLITE_LOCATION:-/opt/glite}/bin/XXX_local_submit_attributes.sh" >> $
↪{bls_opt_req_file}-temp_req_script
  chmod +x ${bls_opt_req_file}-temp_req_script
  ${bls_opt_req_file}-temp_req_script  >> $bls_tmp_file 2> /dev/null
  rm -f ${bls_opt_req_file}-temp_req_script
  rm -f $bls_opt_req_file
fi

# 6. Add specific directives to select queue ($bls_opt_queue) and
#    MPI node request ($bls_opt_mpinodes)

# 7. Add directives to transfer and rename input and output files.
#    These are stored as
#    $bls_inputsand_local_0...$bls_inputsand_local_n-1
#    $bls_inputsand_remote_0...$bls_inputsand_remote_n-1
#    $bls_outputsand_local_0...$bls_outputsand_local_n-1
#    $bls_outputsand_remote_0...$bls_outputsand_remote_n-1
#
#    Two shell functions can help here.
#    a:
#      bls_fl_subst_and_accumulate inputsand "@@F_REMOTE/@@F_LOCAL" "sep"
```

(continues on next page)

(continued from previous page)

```
#      bls_fl_subst_and_accumulate outputsand "@@F_REMOTE/@@F_LOCAL" "sep"
#      fill $bls_fl_subst_and_accumulate_result with a list of "sep"
#      separated strings formatted as shown in the second argument.
#      The submit node full file path is substituted to @@F_LOCAL
#      and the worker node path relative to the initial working dir
#      is substituted to @@F_REMOTE.
#    b:
#      bls_fl_subst_and_dump inputsand "@@F_LOCAL>@@F_REMOTE" $bls_tmp_file
#      bls_fl_subst_and_dump outputsand "@@F_LOCAL<@@F_REMOTE" $bls_tmp_file
#      append to $bls_tmp_file a line for each input and output file,
#      where @@F_REMOTE and @@F_LOCAL are substituted as above.

# 8. Append job wrapper as a shell script to $bls_tmp_file
bls_add_job_wrapper

# 9. Send the submit file $bls_tmp_file to the batch system and
#    try making sure it doesn't get lost.

# 10. Echo to STDOUT the unique job ID to be used by subsequent scripts
#     (with BLAHP_JOBID_PREFIX) and wrap up. The job ID must be
#     properly understood by subsequent commands.
echo "BLAHP_JOBID_PREFIXXXX?$jobID"
bls_wrap_up_submit
exit $retcode
```

6.3.2 XXX_hold.sh script

Suspend the execution of a job.

SYNOPSIS:

```
XXX_hold.sh <job identifier>
```

The job identifier must be the same string returned by the submit script (without the leading BLAHP_JOBID_PREFIX). Any leading part up to the first slash '/' will be ignored by the script.

RETURN VALUE: Termination code is zero on success, nonzero on error.

6.3.3 XXX_resume.sh script

Resume the execution of a (previously suspended) job.

SYNOPSIS:

```
XXX_resume.sh <job identifier>
```

The job identifier must be the same string returned by the submit script (without the leading BLAHP_JOBID_PREFIX). Any leading part up to the first slash '/' will be ignored by the script.

RETURN VALUE: Termination code is zero on success, nonzero on error.

6.3.4 XXX_status.sh script

Get current status of a job.

SYNOPSIS:

```
XXX_status.sh [-w] [-n] <job identifier>
```

The job identifier must be the same string returned by the submit script (without the leading BLAHP_JOBID_PREFIX). Any leading part up to the first slash '/' will be ignored by the script.

- `[-n]`: Option specific to the CREAM caller. Return the port used by the BLParser (see description below) to communicate with the CREAM service
- `[-w]`: Return the worker node where a job is running as an attribute of the output classad (WorkerNode ="host.domain")

RETURN VALUE: The script must return a string-formatted classad (see http://www.cs.wisc.edu/condor/classad/refman/ for the complete reference on classad syntax) containing at least the following attributes:

-
  ```
  BatchjobId = <jobId without the leading batch system name>
  ```

-
  ```
  JobStatus = <status_code>
  ```
  possible value for the status code are 1 = IDLE, 2 = RUNNING, 3 = REMOVED, 4 = COMPLETED, 5 = HELD

-
  ```
  ExitCode = <code>
  ```
  only for COMPLETED jobs

Example of status script results:

- job queued
  ```
  [ BatchjobId = "26526.atlfarm006.mi.infn.it"; JobStatus = 1 ]
  ```
- job completed
  ```
  [ ExitCode = 0; BatchjobId = "26526.atlfarm006.mi.infn.it"; JobStatus = 4 ]
  ```

6.3.5 XXX_cancel.sh script

Remove a job from the batch system queue.

SYNOPSIS:

```
XXX_cancel.sh <job identifier>
```

The job identifier must be the same string returned by the submit script (without the leading BLAHP_JOBID_PREFIX). Any leading part up to the first slash '/' will be ignored by the script.

RETURN VALUE: Termination code is zero on success, nonzero on error.

6.4 BLAH forward requirements to the local batch system

The user can set some requirements to be forwarded to the local batch system, by using the attribute `CERequirements`, as defined above, in the `blah_job_submit` command. This can be achieved both with direct submission to the CREAM CE and with submission to the CE via the WMS, as explained in the following:

- direct submission to CREAM -> the attributes to be forwarded are specified in the .jdl `CERequirements` attribute and are the ones of the GlueSchema in use

- submission to a CE via WMS -> the CERequirements attribute for `blah_job_submit` is filled taking into account the value of the job JDL Requirements expression and what is specified as CeForwardParameters in the WMS conf file (workloadmanager section). Also in this case the parameters to be forwarded are chosen from the GlueSchema in use

6.5 BLAH Commands syntax and semantics

6.5.1 BLAH Commands

The following list of commands represents the set of commands required for interaction with the BLAHP server, interfacing to a given Local Resource Management system. This is based on the minimum set of commands used in the original GAHP (v1.0.0) specification removing commands that are specific to the operation of the GRAM protocol (INITIALIZE_FROM_FILE, GASS_SERVER_INIT, GRAM_CALLBACK_ALLOW, GRAM_JOB_CALLBACK_REGISTER, GRAM_PING). The JOB_SIGNAL command may be initially left unimplemented for some of the batch systems (and in that case will return an error `E` state and will not be returned by COMMANDS).

- BLAH_JOB_CANCEL
- BLAH_JOB_SIGNAL
- BLAH_JOB_HOLD
- BLAH_JOB_REFRESH_PROXY
- BLAH_JOB_RESUME
- BLAH_JOB_STATUS
- BLAH_JOB_STATUS_ALL
- BLAH_JOB_STATUS_SELECT
- BLAH_JOB_SUBMIT
- BLAH_SET_GLEXEC_DN
- BLAH_SET_GLEXEC_OFF
- COMMANDS
- CACHE_PROXY_FROM_FILE
- QUIT
- RESULTS
- USE_CACHED_PROXY
- UNCACHE_PROXY
- VERSION

Optionally, the following two commands may also be implemented:

- ASYNC_MODE_ON
- ASYNC_MODE_OFF

6.5.2 BLAHP Commands structure

Conventions and Terms used

Below are definitions for the terms used in the sections to follow:

- `<CRLF>`The characters carriage return and line feed (in that order), or solely the line feed character.
- `<SP>`The space character.
- `line` A sequence of ASCII characters ending with a `<SP>`
- `Request Line` A request for action on the part of the BLAHP server.
- `Return Line` A line immediately returned by the BLAHP server upon receiving a Request Line.
- `Result Line` A line sent by the BLAHP server in response to a RESULTS request, which communicates the results of a previous asynchronous command Request.
- `S:` and `R:` In the Example sections for the commands below, the prefix “S: ” is used to signify what the client sends to the BLAHP server. The prefix “R: ” is used to signify what the client receives from the BLAHP server. Note that the “S: ” or “R: ” should not actually be sent or received.

Commands structure

BLAHP commands consist of three parts:

- Request Line
- Return Line
- Result Line

Each of these “Lines” consists of a variable length character string ending with the character sequence <CRLF>.

A Request Line is a request from the client for action on the part of the BLAHP server. Each Request Line consists of a command code followed by argument field(s). Command codes are a string of alphabetic characters. Upper and lower case alphabetic characters are to be treated identically with respect to command codes. Thus, any of the following may represent the blah_job_submit command:

- blah_job_submit
- Blah_Job_Submit
- blAh_joB_suBMit
- BLAH_JOB_SUBMIT

In contrast, the argument fields of a Request Line are *case sensitive*.

The Return Line is always generated by the server as an immediate response to a Request Line. The first character of a Return Line will contain one the following characters:

- S - for Success
- F - for Failure
- E - for a syntax or parse Error

Any Request Line which contains an unrecognized or unsupported command, or a command with an insufficient number of arguments, will generate an “E” response.

The Result Line is used to support commands that would otherwise block. Any BLAHP command which may require the implementation to block on network communication require a “request id” as part of the Request Line. For such commands, the Result Line just communicates if the request has been successfully parsed and queued for service by the BLAHP server. At this point, the BLAHP server would typically dispatch a new thread to actually service the request. Once the request has completed, the dispatched thread should create a Result Line and enqueue it until the client issues a RESULT command.

Transparency

Arguments on a particular Line (be it Request, Return, or Result) are typically separated by a <SP>. In the event that a string argument needs to contain a <SP> within the string itself, it may be escaped by placing a backslash (“") in front of the <SP> character. Thus, the character sequence “\ ” (no quotes) must not be treated as a separator between arguments, but instead as a space character within a string argument.

Sequence of Events

Upon startup, the BLAHP server should output to stdout a banner string which is identical to the output from the VERSION command without the beginning “S ” sequence (see example below). Next, the BLAHP server should wait for a complete Request Line from the client (e.g. stdin). The server is to take no action until a Request Line sequence is received.

Example:

```
R: $GahpVersion: x.y.z Feb 31 2004 INFN\ Blahpd $
  S: COMMANDS
  R: S COMMANDS BLAH_JOB_CANCEL BLAH_JOB_SIGNAL BLAH_JOB_STATUS BLAH_JOB_SUBMIT␣
↪COMMANDS QUIT RESULTS VERSION
  S: VERSION
  R: S $GahpVersion: x.y.z Feb 31 2004 INFN\ Blahpd $
                (other commands)
  S: QUIT
  R: S
```

6.5.3 BLAH Commands syntax

This section contains the syntax for the Request, Return, and Result line for each of the following commands:

- `COMMANDS`: List all the commands from this protocol specification which are implemented by this BLAHP server.

  Request Line: COMMANDS <CRLF>

  Return Line: S <SP> <COMMAND 1> <SP> <COMMAND 2> <SP> . . . <CRLF>

  Result Line: None.

- `VERSION`: Return the version string for this BLAHP. The version string follows a specified format (see below). Ideally, the version entire version string, including the starting and ending dollar sign ($) delimiters, should be a literal string in the text of the BLAHP server executable. This way, the Unix/RCS “ident” command can produce the version string. The version returned should correspond to the version of the protocol supported.

  Request Line: VERSION <CRLF>

  Return Line: S <SP> $GahpVesion: <SP> <major>.<minor>.<subminor> <SP> <build-month> <SP> <build-day-of-month> <SP> <build-year> <SP> <general-descrip> <SP>$ <CRLF>

  – major.minor.subminor = for this version of the protocol, use version 1.0.0.

  - build-month = string with the month abbreviation when this BLAHP server was built or released. Permitted values are: “Jan”, “Feb”, “Mar”, “Apr”, “May”, “Jun”, “Jul”, “Aug”, “Sep”, “Oct”, “Nov”, and “Dec”.
  - build-day-of-month = day of the month when BLAHP server was built or released; an integer between 1 and 31 inclusive.
  - build-year = four digit integer specifying the year in which the BLAHP server was built or released.
  - general-descrip = a string identifying a particular BLAHP server implementation.

  Result Line: None.

  Example:

```
S: VERSION R: S $GahpVersion: x.y.z Feb 31 2004 INFN\ Blahpd $
```

- `QUIT`: Free any/all system resources (close all sockets, etc) and terminate as quickly as possible.

  Request Line: QUIT <CRLF>

  Return Line: S <CRLF>

  Immediately afterwards, the command pipe should be closed and the BLAHP server should terminate.

  Result Line: None.

- `RESULTS`: Display all of the Result Lines which have been queued since the last RESULTS command was issued. Upon success, the first return line specifies the number of subsequent Result Lines which will be displayed. Then each result line appears (one per line) – each starts with the request ID which corresponds to the request ID supplied when the corresponding command was submitted. The exact format of the Result Line varies based upon which corresponding Request command was issued.

  IMPORTANT: Result Lines must be displayed in the *exact order* in which they were queued!!! In other words, the Result Lines displayed must be sorted in the order by which they were placed into the BLAHP’s result line queue, from earliest to most recent.

  Request Line: RESULTS <crlf>

  Return Line(s): S <SP><num-of-subsequent-result-lines> <CRLF><reqid> <SP> … <CRLF> <reqid> <SP> … <CRLF>…

  reqid = integer Request ID, set to the value specified in the corresponding Request Line.

  Result Line: None.

  Example:

```
S: RESULTS
  R: S 1
  R: 100 0
```

- `ASYNC_MODE_ON`: Enable Asynchronous notification when the BLAHP server has results pending for a client. This is most useful for clients that do not want to periodically poll the BLAHP server with a RESULTS command. When asynchronous notification mode is active, the GAHP server will print out an `R` (without the quotes) on column one when the ‘RESULTS’ command would return one or more lines. The `R` is printed only once between successive `RESULTS` commands. The `R` is also guaranteed to only appear in between atomic return lines; the `R` will not interrupt another command’s output.

  If there are already pending results when the asynchronous results available mode is activated, no indication of the presence of those results will be given. A GAHP server is permitted to only consider changes to it’s result queue for additions after the ASYNC_MODE_ON command has successfully completed. GAHP clients should issue a `RESULTS` command immediately after enabling asynchronous notification, to ensure that any results that may have been added to the queue during the processing of the ASYNC_MODE_ON command are accounted for.

Request Line: ASYNC_MODE_ON <CRLF>

Return Line: S <CRLF> Immediately afterwards, the client should be prepared to handle an R <CRLF> appearing in the output of the GAHP server.

Result Line: None.

Example:

```
S: ASYNC_MODE_ON
R: S
S: BLAH_JOB_CANCEL 00001 123.bbq.mi.infn.it
R: S
S: BLAH_JOB_CANCEL 00002 124.bbq.mi.infn.it
R: S
R: R
S: RESULTS
R: S 2
R: 00001 0
R: 00002 0
R: 00002 0
R: 00002 0
R: 00002 0
R: 00002 0
R: 00002 0
R: 00002 0
R: 00002 0
R: 00002 0
R: 00002 0
R: 00002 0
R: 00002 0
R: 00002 0
R: 00002 0
R: 00002 0
R: 00002 0
```

Note that you are NOT guaranteed that the `R` will not appear between the dispatching of a command and the return line(s) of that command; the GAHP server only guarantees that the `R` will not interrupt an in-progress return. The following is also a legal example:

```
S: ASYNC_MODE_ON
      R: S
      S: BLAH_JOB_CANCEL 00001 123.bbq.mi.infn.it
      R: S
      S: BLAH_JOB_CANCEL 00002 124.bbq.mi.infn.it
      R: R
      R: S
      S: RESULTS
      R: S 2
      R: 00001 0
      R: 00002 0
```

- `ASYNC_MODE_OFF`: Disable asynchronous results-available notification. In this mode, the only way to discover available results is to poll with the RESULTS command. This mode is the default. Asynchronous mode can be enabled with the ASYNC_MODE_ON command.

  Request Line: ASYNC_MODE_OFF <CRLF>

Return Line: S <CRLF>

Results Line: None

Example:

```
S: ASYNC_MODE_OFF
R: S
```

6.5.4 BLAH_JOB_SUBMIT

Submit a job request to a specified queue (specified in the submit classad). This will cause the job to be submitted to the batch system.

Request Line: BLAH_JOB_SUBMIT <SP> <reqid> <SP> <submit classad> <CRLF>

- reqid = non-zero integer Request ID
- submit classad = valid submit description for the job, in string representation. See paragraph 3.0 for a description of the format. Here's a list of supported attributes with a brief description.
  - "Cmd": Full path of the executable in the local filesystem
  - "Args": List of individual arguments (no '/bin/sh' convention on argument separation, but separate arguments) for the executable
  - "In": Full path in the local filesystem where the standard input for the executable is found
  - "Out": Full path in the local filesystem where the standard output of the executable will be stored (at job completion).
  - "Err": Full path in the local filesystem where the standard error of the executable will be stored (at job completion).
  - "X509UserProxy": Full path wherethe proxy certificate is stored.
  - "Env": Semicolon-separated list of environment variables of the form:

    ```
    <parameter> = <value>
    ```

  - "Stagecmd": Sets if the executable of the job must be copied on the WorkerNode: can be "TRUE" or "FALSE".
  - "Queue": Queue in the local batch system where the job must be enqueued.
  - "Gridtype": String indicating the underlying local batch system (currently "pbs" and "lsf" supported).
  - "uniquejobid": unique name identifier for the final job to be submitted to the local batch system.
  - "NodeNumber": number of nodes to be reserved for an MPI job. It gets translated into the LRMS command qsub "-l nodes=<n>" for pbs or into "-n <n>" for lsf. For Condor this attribute is not supported, therefore it gets ignored.
  - "CERequirements": string containing the requirements to be forwarded to the local batch system
  - "TransferInput = file1,file2,file…": comma-delimited list of all the files to be transferred into the working directory for the job before the job is started. Only the transfer of files is available. The transfer of subdirectories is not supported.
  - "TransferOutput = file1,file2,file…": an explicit list of output files to be transferred back from the temporary working directory on the execute machine to the submit machine (where BLAHPD is running). Only the standard output and standard error files, are transferred back by default if requested.

   - "TransferOutputRemaps = name1=newname1;name2=newname2;name=newname…": This attribute specifies the name (and optionally the complete path) to use when downloading output files from the completed job. *Note that the mappings are separated by semicolons*. Normally, output files are transferred back to the initial working directory with the same name they had in the execution directory. This gives you the option to save them with a different path or name. If you specify a relative path, the final path will be relative to the job's initial working directory (the directory specified in the 'Iwd' attribute, or, if this is missing, the current working directory of BLAHPD)

Return Line:

```
<result> <CRLF>
```

- result = the character "S" (no quotes) for successful submission of the request (meaning that the request is now pending), or an "E" for error on the parse of the request or its arguments (e.g. an unrecognized or unsupported command, or for missing or malformed arguments).

Result Lines: <reqid> <sp> <result-code> <sp> <error-string> <sp> <job_local_id> <crlf>

- reqid = integer Request ID, set to the value specified in the corresponding Request Line.
- result-code = integer equal to 0 on success, or an error code
- error-string = description of error
- job_local_id = on success, a string representing a unique identifier for the job. This identifier must not be bound to this BLAHP server, but instead must be allowed to be used in subsequent BLAHP server instantiations. For instance, the job_local_id must be implemented in such a fashion that the following sequence of events by the caller must be permissible:
   - step a: issue a BLAH_JOB_SUBMIT command
   - step b: read the job_local_id in the result line
   - step c: store the job_local_id persistently
   - step d: subsequently kill and restart the BLAHP server process
   - step e: issue a BLAH_JOB_CANCEL command, passing it the stored job_local_id value obtained in step (b).

Example:

```
S: BLAH_JOB_SUBMIT 2 [\ Cmd\ =\ "/usr/bin/test.sh";\ Args\ =\ "'X=3:Y=2'";
   \ Env\ =\ "VAR1=56568";\ In\ =\ "/dev/null";\ Out\ =\ "/home/StdOutput";
   \ Err\ =\ "/home/error";\ x509userproxy\ =\ "/home/123.proxy";\ Stagecmd
   \ =\ TRUE;\ Queue\ =\ "short";\ GridType\ =\ "pbs";\ ]'
R: S
S: RESULTS
R: 2 0 No\ error pbs/20051012/2957
```

6.5.5 BLAH_JOB_CANCEL

This function removes an IDLE job request, or kill all processes associated with a RUNNING job, releasing any associated resources.

Request Line: BLAH_JOB_CANCEL <sp> <reqid> <sp> <job_local_id> <CRLF>

- reqid = non-zero integer Request ID
- job_local_id = job_local_id (as returned from BLAH_JOB_SUBMIT) of the job to be canceled.

Return Line: <result> <crlf>

- result = the character “S” (no quotes) for successful submission of the request (meaning that the request is now pending), or an “E” for error on the parse of the request or its arguments (e.g. an unrecognized or unsupported command, or for missing or malformed arguments).

Result Line: <reqid> <sp> <result-code> <sp> <error-string> <CRLF>

- reqid = integer Request ID, set to the value specified in the corresponding Request Line.
- result-code = integer equal to 0 on success, or an error code
- error-string = description of error

Example:

```
S: BLAH_JOB_CANCEL 1 pbs/20051012/2957.grid001.mi.infn.it
R: S
R: R
S: RESULTS
R: S 1
R: 1 0 No\ error
```

6.5.6 BLAH_JOB_STATUS

Query and report the current status of a submitted job.

Request Line: BLAH_JOB_STATUS <sp> <reqid> <sp> <job_local_id> <CRLF>

- reqid = non-zero integer Request ID
- job_local_id = job_local_id (as returned from BLAH_JOB_SUBMIT) of the job whose status is desired.

Return Line: <result> <CRLF>

- result = the character “S” (no quotes) for successful submission of the request (meaning that the request is now pending), or an “E” for error on the parse of the request or its arguments (e.g. an unrecognized or unsupported command, or for missing or malformed arguments).

Result Line: <reqid> <sp> <result-code> <sp> <error-string> <sp> <job_status> <sp> <result-classad> <CRLF>

- reqid = integer Request ID, set to the value specified in the corresponding Request Line.
- result-code = integer equal to 0 on success, or an error code
- error-string = description of error
- job_status = if the result_code is 0 (success), then job_status is set to an integer based upon the status of the job as follows:
  - 1 IDLE (job is waiting on the batch system queue)
  - 2 RUNNING (job is executing on a worker node)
  - 3 REMOVED (job was successfully cancelled)
  - 4 COMPLETED (job completed its execution on the batch system)
  - 5 HELD (job execution is suspended; job is still in the batch system queue)
- result-classad = Aggregate information about the job status. The classad format can vary with the local batch system. Typically, the following attributes are defined:
  - JobStatus : job status - same codes as described above
  - BatchjobId : Job ID as known to the local batch system

- ExitCode : Termination code - only for finished jobs
- ExitReason : Exit reason, if available - only for finished jobs
- WorkerNode : When available, FQDN of the worker node - only for running jobs

Example:

```
S: BLAH_JOB_STATUS 1 pbs/20051012/2958.grid001.mi.infn.it
R: S
R: R
S: RESULTS
R: S 1
R: 1 0 No\ Error 2 [\ BatchjobId\ =\ "2958.grid001.mi.infn.it";
   \ JobStatus\ =\ 2;\ WorkerNode\ =\ "\ grid001.mi.infn.it"\ ]
```

6.5.7 BLAH_JOB_STATUS_ALL

This command is only available if the BLAH local job registry file is configured in the BLAH config file (job_registry attribute) and supported by the active batch system. Query and report the current status of all jobs managed by the BLAH server.

Request Line: BLAH_JOB_STATUS_ALL <sp> <reqid> <CRLF>

- reqid = non-zero integer Request ID

Return Line: <result> <CRLF>

- result = the character "S" (no quotes) for successful submission of the request (meaning that the request is now pending), or an "E" for error on the parse of the request or its arguments (e.g. an unrecognized or unsupported command, or for missing or malformed arguments).

Result Line: <reqid> <sp> <result-code> <sp> <error-string> <sp> <result-classad> <CRLF>

- reqid = integer Request ID, set to the value specified in the corresponding Request Line.
- result-code = integer equal to 0 on success, or an error code
- error-string = description of error
- result-classad = List of classads containing aggregate information about the job status. In addition to the attributes defined for the output of the BLAH_JOB_STATUS command, as the status info comes from a local cache, the following attributes may also be present:
  - BlahJobId: Job ID as known to the BLAH layer. Typically a decorated form of BatchJobId
  - CreateTime: Seconds since the Unix epoch at the time the first job info was inserted into the cache.
  - ModifiedTime: Seconds since the Unix epoch when the most recent modification of the job info occurred.

6.5.8 BLAH_JOB_STATUS_SELECT

This command is only available if the BLAH local job registry file is configured in the BLAH config file (job_registry attribute) and supported by the active batch system. Query and report the current status of all jobs managed by the BLAH server.

Request Line: BLAH_JOB_STATUS_SELECT <sp> <reqid> <sp> <selection expression> <CRLF>

- reqid = non-zero integer Request ID
- selection expression = Classad expression defining select requirements on the returned job ads (e.g.: JobStatus ==2).

Return Line: <result> <crlf>

- result = the character "S" (no quotes) for successful submission of the request (meaning that the request is now pending), or an "E" for error on the parse of the request or its arguments (e.g. an unrecognized or unsupported command, or for missing or malformed arguments).

Result Line: <reqid> <sp> <result-code> <sp> <error-string> <sp> <result-classad> <CRLF>

- reqid = integer Request ID, set to the value specified in the corresponding Request Line.
- result-code = integer equal to 0 on success, or an error code
- error-string = description of error
- result-classad = List of classads containing aggregate information about the status of the jobs that satisfy the selection requirement. The format is the same as for the BLAH_JOB_STATUS_ALL command.

6.5.9 BLAH_JOB_SIGNAL

Send a signal (if possible) to a specified job. This has to be in the RUNNING status.

Request Line: BLAH_JOB_SIGNAL <sp> <reqid> <sp> <job_local_id> <sp> <signal> <CRLF>

- reqid = non-zero integer Request ID
- job_local_id = job_local_id (as returned from BLAH_JOB_SUBMIT) of the job whose status is desired.
- signal = an integer with the signal to send

Return Line:<result> <CRLF>

- result = the character "S" (no quotes) for successful submission of the request (meaning that the request is now pending), or an "E" for error on the parse of the request or its arguments (e.g. an unrecognized or unsupported command, or for missing or malformed arguments).

Result Line: <reqid> <sp> <result-code> <sp> <error-string> <sp> <job_status> <CRLF>

- reqid = integer Request ID, set to the value specified in the corresponding Request Line.
- result-code = integer equal to 0 on success, or an error code
- error-string = description of error
- job_status = if the result_code is 0 (success), then job_status is set to an integer based upon the status of the job as follows (compare above):
  - 1.IDLE
  - 2. RUNNING
  - 3. REMOVED
  - 4. COMPLETED
  - 5. HELD

6.5.10 BLAH_JOB_REFRESH_PROXY

Renew the proxy of an already submitted job. The job has to be in IDLE, RUNNING or HELD status.

Request Line: BLAH_JOB_REFRESH_PROXY <sp> <reqid> <sp> <job_local_id> <sp> <proxy_file> <CRLF>

- reqid = non-zero integer Request ID
- job_local_id = job_local_id (as returned from BLAH_JOB_SUBMIT) of the job whose proxy has to be renewed.

- proxy_file = path to the fresh proxy file.

Return Line: <result> <CRLF>

- result = the character "S" (no quotes) for successful submission of the request (meaning that the request is now pending), or an "E" for error on the parse of the request or its arguments (e.g. an unrecognized or unsupported command, or for missing or malformed arguments).

Result Line: <reqid> <sp> <result-code> <sp> <error-string> <crlf>

- reqid = integer Request ID, set to the value specified in the corresponding Request Line.
- result-code = integer equal to 0 on success, or an error code
- error-string = description of error

Example:

```
S: BLAH_JOB_REFRESH 1 123.proxy
R: S
R: R
S: RESULTS
R: S 1
R: 1 0 No\ Error
```

6.5.11 BLAH_JOB_HOLD

This function always puts an IDLE job request in a HELD status. If the job is already running RUNNING it can be HELD too, depending whether the underlying batch system supports this feature.

Request Line: BLAH_JOB_HOLD <sp> <reqid> <sp> <job_local_id> <crlf>

- reqid = non-zero integer Request ID
- job_local_id = job_local_id (as returned from BLAH_JOB_SUBMIT) of the job to be canceled.

Return Line: <result> <crlf>

- result = the character "S" (no quotes) for successful submission of the request (meaning that the request is now pending), or an "E" for error on the parse of the request or its arguments (e.g. an unrecognized or unsupported command, or for missing or malformed arguments).

Result Line: <reqid> <sp> <result-code> <sp> <error-string> <crlf>

- reqid = integer Request ID, set to the value specified in the corresponding Request Line.
- result-code = integer equal to 0 on success, or an error code
- error-string = description of error

Example:

```
S: BLAH_JOB_HOLD 1 pbs/20051012/2957.grid001.mi.infn.it
R: S
R: R
S: RESULTS
R: S 1
R: 1 0 No\ error
```

6.5.12 BLAH_JOB_RESUME

This function puts an HELD job request in the status it was before the holding action.

Request Line: BLAH_JOB_RESUME <sp> <reqid> <sp> <job_local_id> <crlf>

- reqid = non-zero integer Request ID
- job_local_id = job_local_id (as returned from BLAH_JOB_SUBMIT) of the job to be canceled.

Return Line: <result> <crlf>

- result = the character "S" (no quotes) for successful submission of the request (meaning that the request is now pending), or an "E" for error on the parse of the request or its arguments (e.g. an unrecognized or unsupported command, or for missing or malformed arguments).

Result Line: <reqid> <sp> <result-code> <sp> <error-string> <crlf>

- reqid = integer Request ID, set to the value specified in the corresponding Request Line.
- result-code = integer equal to 0 on success, or an error code
- error-string = description of error

Example:

```
S: BLAH_JOB_RESUME 1 pbs/20051012/2957.grid001.mi.infn.it
R: S
R: R
S: RESULTS
R: S 1
R: 1 0 No\ error
```

6.5.13 BLAH_SET_GLEXEC_DN

This function enables local user mapping via GLEXEC for all subsequent actions. The call is synchronous and the effect is immediate.

Request Line: BLAH_SET_GLEXEC_DN <sp> <user_proxy> <sp> <ssl_client_certificate> <sp> <not_limited> <crlf>

- user_proxy Path to a valid GSI proxy file. This will be optionally (when not_limited == 0) turned into a limited (delegated) proxy and be made available via GLEXEC to the user job. (The delegated proxy path is set in the GLEXEC_SOURCE_PROXY enviroment variable prior to the GLEXEC call).
- ssl_client_certificate Path to a valid GSI proxy file. Can be the same as user_proxy. This is the client certificate that is used for access to GLEXEC and local user mapping (it is set in the GLEXEC_CLIENT_CERT environment variable prior to the GLEXEC call).
- not_limited If this argument is set to a numeric value not equal to zero, the supplied proxy will not be limited (delegated).

Return Line: one of the following

- S <sp> <action description> <crlf> - in case of success
- F <crlf> - in case of failure

Result Line: None.

Example:

```
S: BLAH_SET_GLEXEC_DN /path/test.proxy /path/test.proxy 0
R: S Glexec\ mode\ on
```

6.5.14 BLAH_SET_GLEXEC_OFF

This command unconditionally disables user mapping via GLEXEC for all subsequent actions. The call is synchronous and the effect is immediate.

Request Line: BLAH_SET_GLEXEC_OFF <crlf>

- reqid = integer Request ID, set to the value specified in the corresponding Request Line.

Return Line: one of the following

- S <sp> <action-description> <crlf> - in case of success
- F <crlf> - in case of failure

Result Line: None.

Example:

```
S: BLAH_SET_GLEXEC_OFF
R: S Glexec\ mode\ off
```

6.5.15 CACHE_PROXY_FROM_FILE

This command is only available if enable_glexec_from_condor is set to 'yes' in the BLAHPD config file. Read a GSI proxy from a specified file, and store it under the specified symbolic name. Unlike other GAHPs, and because this function is only used to trigger GLEXEC access which requires access to a proxy file, BLAH does not read and cache the proxy contents but only the file name. Changes to the file will affect the use of the stored proxy. The proxy name caching is lost when the BLAHP server terminates. The intent of this command is to allow the BLAHP server to capture and use different proxy files for different jobs. Other commands (e.g. by USE_CACHED_PROXY) may later be reference the given cached proxy file by its symbolic name. USE_CACHED_PROXY has the side effect of enabling GLEXEC.

Request Line:CACHE_PROXY_FROM_FILE <sp> <id> <sp> <path-to-local-proxy-file> <crlf>

- <id> = a symbolic name which will be used to reference the given proxy.
- <path-to-local-proxy-file> = a fully-qualified pathname to a file local to the BLAHP server which contains a valid GSI proxied certificate.

Return Line: one of the following:

- S <sp> <action-description> <crlf>
- F <sp> <error-string> <crlf>Upon success, use the "S" version; if not recognized, use the "F" version.
  - error-string = brief string description of the error, appropriate for reporting to a human end-user.

Result Line:None.

Example:

```
S: CACHE_PROXY_FROM_FILE 1 /path/test.proxy
R: S Proxy\ cached
```

6.5.16 USE_CACHED_PROXY

This command is only available if enable_glexec_from_condor is set to 'yes' in the BLAHPD config file. Sets the proxy previously cached under the specified name, as the "active" proxy. Enable user mapping via GLEXEC for all subsequent actions. The active proxy will be used for GLEXEC user mapping, it will remain active until it is changed via the next invocation of USE_CACHED_PROXY. The proxy must have been previously cached (under the specified name) using CACHE_PROXY_FROM_FILE command. This command allows the BLAHP server to simultaneously act on behalf of two or more jobs that require

Request Line: USE_CACHED_PROXY <sp> <id> <crlf> <id> = the symbolic name of the proxy to use

Return Line: One of the following:

- S <sp> <action-description> <crlf>
- F <sp> <error-string> <crlf> Upon success, use the "S" version; if not recognized, use the "F" version.
  - error-string = brief string description of the error, appropriate for reporting to a human end-user.

Result Line: None.

Example:

```
S: USE_CACHED_PROXY 1
R: S Glexec\ proxy\ set\ to\ /path/test.proxy\ (dir\ IWGRP\ bit\ already\ on)
```

6.6 BLParser

To obtain efficient access and prompt update to the current status of BLAH active jobs, a 'log parser' daemon was developed for LSF and PBS. The submit and status scripts can make optional use of the batch log parser to increase their efficiency.

Here's a description of the semantics of the batch log parser (BLParser).

The BLParser listens on a network socket and replies to a set of mandatory and a set of optional queries (optional queries can be used for info and debug).

Here is the list of mandatory queries and their replies:

- BLAHJOB/<blahjob-id> <lrms-jobid>
- <date-yyyymmdd>/<lrms-jobid> a classad like this:

  ```
  [
    BatchJobId=<lrms-jobid>
    Workernode=<workernode>
    JobStatus=<jobstatus>
    LRMSSubmissionTime=<submission time>
    LRMSStartRunningTime=<start time>
    LRMSCompletedTime=<finish time>
    ExitReason=<exitreason>
    ExitCode=<exit code>
  ]/pr_removal
  ```

  where pr_removal is a flag that told the status script if the proxy file has to be removed (when job is killed or finished) or not.

- CREAMPORT => port where cream can connect to the parser.

- TEST => print Y<lrmsname> (e.g. YLSF YPBS). This query is used to know if a parser is active. This is useful when in config file is specified more than one parser to try: if the first does not reply to the TEST query the next one is tried.

These are optional queries and their replies:

- HELP => print command list
- VERSION => print version
- TOTAL => print total number of jobs cached in the parser
- LISTALL => print jobid of all jobs cached in the parser
- LISTF[/<first-n-jobid>] => print first n jobid
- LISTL[/<last-n-jobid>] => print last n jobid

The BLParser includes an option to notify to a client (typically the CREAM service) every state change in CREAM-managed jobs. CREAM sends to the parser a string like this:

```
STARTNOTIFY/<date-yyyymmdd>
```

atfer that string the parser sends back to cream every state change for cream job since the date contained in the string with lines like this:

```
NTFDATE/<classad>
```

where <classad> is a classad like the one sent in reply to <date-yyyymmdd>/<lrms-jobid> query.

After that parser will sent to cream every new state change for cream jobs. Cream can change the default prefix (cream_) used to identify tracked jobs among all other jobs with this command:

```
CREAMFILTER/<creamfilter>
```

where <creamfilter> should be a string like crXXX_ (XXX can be any character [a-z],[A-Z],[0-9]). This command has to sent before STARTNOTIFY.

There is a startup script for the BLParser that reads all the parameters from a config file. It is possible to set the debug level, the debug log file, the log file location; it is also possible to start different parsers listening on different ports with the same startup script.

6.7 New Parser Scenario

The old BLParser is now splitted in two different daemons:

- BUpdater (that is specific for each batch system:there are BUpdaterLSF, BUpdaterPBS and BUpdaterCondor)
- BNotifier that is the same for all batch systems.

Both daemons are now automatically launched and controlled by the blahpd daemon so there is no need for a startup script. The main differences between this approach and the old one are that the daemons run as non privileged user and that the informations about jobs are collected using the batch system native command (as bhist, bjobs, qstat…) and are saved in a db file on disk called job registry. These informations are asynchronously notified to CREAM by BNotifier daemon.

It is possible to configure some parameter in the blah.config file:

- _bupdater_path_ and _bnotifier_path_: if this is set to the bupdater (or bnotifier) complete path (e.g. /opt/glite/bin/BUpdaterLSF) blahpd takes care of starting and periodically check the status of the daemon. If it is not set the daemons should be started by hand.

- _bupdater_pidfile_ and _bnotifier_pidfile_: this are used to check the daemons status by blahpd (e.g. /var/tmp/bupdater.pid).
- _job_registry_: is the complete path to the job registry location (/var/tmp/job_registry.db).
- _bupdater_debug_level_: set the debug level (1,2,3)
- _bupdater_debug_logfile_: set the location of the debug log file
- _async_notification_host_: hostname where the notification should be sent
- _async_notification_port_: port where to send notifications
- _purge_interval_: how old (in seconds) a job registry entry has to be before it is purged.
- _finalstate_query_interval_: the query that search for finished jobs is executed every finalstate_query_interval seconds (default:30)
- _alldone_interval_: after that interval an unseen job is set as done (status = 4) and exitstatus = -1 (default:600)
- _bupdater_loop_interval_: the updater main lop is executed every bupdater_loop_interval seconds (default:5)
- _blah_graceful_kill_mappable_cmd_: Command to use when terminating processes via a user mapping tool (glexec or SUDO). Default is /bin/kill
- _blah_graceful_kill_timeout_: Total number of seconds a parent process will wait for a child to terminate after sending it the first termination (SIGTERM) signal. During this time, and until the child process terminates, the parent will send SIGTERM to che child once a second before blah_graceful_kill_timeout/2 seconds have elapsed, then it will send SIGKILL once a second. Defaults to 20 seconds.
- _blah_children_restart_interval_: Mininum time in seconds that has to elapse before blahpd tries restarting one of the children processes it controls. Defaults to 150 seconds.
- _blah_require_proxy_on_submit_: Set to boolean 'true' to generate an error when the X509Proxy attribute is missing from the job description in the BLAH_JOB_SUBMIT command. Set to boolean 'false' if job submission without a valid proxy is allowed. Defaults to 'false'.
- _blah_enable_glexec_from_condor_: Set to boolean 'true' to make blahpd recognize the following three commands: CACHE_PROXY_FROM_FILE, USE_CACHED_PROXY, UNCACHE_PROXY. User mapping via GLEXEC will be automatically enabled when the USE_CACHED_PROXY command is received. This is to allow user switching via glexec based on the active proxy for a given user, as cached by the Condor gridmanager. Defaults to 'false'.
- _blah_disable_wn_proxy_renewal_: When set to boolean 'true', use of the BPRclient/server tool to delegate or send renewed X509 proxies to a job executing node will be entirely disabled. Defaults to 'false'.
- _blah_delegate_renewed_proxies_: When set to boolean 'true', X509 proxies sent to job worker nodes via the BPRclient/server tool will be delegated instead of being copied over the wire. Defaults to 'false' as many Globus tools (including gridftp) will reject access when a limited -and- delegated proxy is presented.
- _blah_graceful_kill_glexecable_cmd_format_

6.8 BLAH Accounting

Log records are written in a daily file specified as

```
<prefix>-YYYMMDD
```

where `<prefix>` is specified in blah.config:

```
BLAHPD_ACCOUNTING_INFO_LOG=/var/log/accounting/blahp.log
```

The format of the record is:

```
"<field>=<value>" [<SP> "<field>=<value>" ...] <LF>
```

thus all values are assumed to have no double quotes inside.

The fields are:

- `timestamp`: record creation date (~ lrms submission time)
- `userDN`: user's certificate distinguished name
- `userFQAN`: user's certificate Fully Qualified Attribute Names (i.e. VO and roles) repeated, one per FQAN
- `ceID`: Computing Element's ID as appearing in the Glue schema
- `jobID`: the Grid job ID (i.e. the one returned to the user by the CE)
- `lrmsID`: the ID of the job in the batch system
- `localUser`: numeric uid of the local account mapped to the remote user
- `clientId`: a job identifier (arbirtary string) provided by the CE at submission time

Accounting can be disabled in BLAH by commenting out the BLAHPD_ACCOUNTING_INFO_LOG line in blah.config.

CHAPTER 7

CREAM monitoring

The CREAM CE site is monitored with the Nagios framework and a set of specific probes. The service can be tested with job submission through WMS or with direct submittion (i.e. using CREAM CLI). The probes developed for the CREAM service must be installed on a User Interface because they use the cream-cli commands to monitor the CREAM ce.

7.1 Requirements

The required packages for the CREAM CE probes are:

- python (version 2.7 or greater)
- python-ldap
- python-suds (version 0.3.5 or greater)
- openssl (version 0.9.8e-12 or greater)
- nagios-submit-conf (version 0.2 or greater)
- python-GridMon (version 1.1.10)

About the last two rpms they can be install using the EGI repository

7.2 CREAM-CE direct job submission metrics and WN probes

The following metrics are a restructured version of the existing ones and provide a better approach for probing a CREAM CE and its WNs:

- `cream_serviceInfo.py` - get CREAM CE service info
- `cream_allowedSubmission.py` - check if the submission to the selected CREAM CE is allowed
- `cream_jobSubmit.py` - submit a job directly to the selected CREAM CE

- `cream_jobOutput.py` - submit a job directly to the selected CREAM CE and retrieve the output-sandbox
- `WN-softver probe` - check middleware version on WN (via cream_jobOutput.py)
- `WN-csh probe` - check if WN has csh (via cream_jobOutput.py)

All of them have been implemented in python and are based on the cream-clicommands. They share the same logic structure and provide useful information about their version, usage (i.e. help) including the options list and their meaning, according to the guide Probes Development. For example:

```
$ ./cream_serviceInfo.py
Usage: cream_serviceInfo.py [options]

cream_serviceInfo.py: error: Specify either option -u URL or option -H HOSTNAME (and -
↪p PORT) or read the help (-h)

$ ./cream_serviceInfo.py --help
Usage: cream_serviceInfo.py [options]

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -H HOSTNAME, --hostname=HOSTNAME
                        The hostname of the CREAM service.
  -p PORT, --port=PORT  The port of the service. [default: none]
  -x PROXY, --proxy=PROXY
                        The proxy path
  -t TIMEOUT, --timeout=TIMEOUT
                        Probe execution time limit. [default:
                        120 sec]
  -v, --verbose         verbose mode [default: False]
  -u URL, --url=URL     The status endpoint URL of the service. Example:
                        https://<host>[:<port>]

$ ./cream_serviceInfo.py --version
cream_serviceInfo v.1.1
```

The interaction with the CREAM CE requires the use of a valid VOMS proxy expressed by the `X509_USER_PROXY` env variable or through the `--proxy` option. All metrics check the existence of the proxy file and calculate the time left. In case of error, the related error message will be thrown:

```
$ ./cream_serviceInfo.py --hostname cream-41.pd.infn.it --port 8443 --proxy /tmp/
↪x509up_u0 --verbose
Proxy file not found or not readable
```

The verbose mode (`--verbose`) could be enabled to each metric. It provides several details about the probe execution itself by highlighting the internal commands:

```
$ ./cream_serviceInfo.py --hostname prod-ce-01.pd.infn.it --port 8443 -x /tmp/x509up_
↪u733 --verbose
executing command: /usr/bin/voms-proxy-info -timeleft
invoking service info
executing command: /usr/bin/glite-ce-service-info prod-ce-01.pd.infn.it:8443
CREAM serviceInfo OK: Service Version    = [1.16.4 - EMI version: 3.15.0-1.el6]
```

In case of mistakes on the selected options or on their values, the probe tries to explain what is wrong. For example the cream_serviceInfo doesn't support the `--queue` option:

```
$ ./cream_serviceInfo.py --hostname prod-ce-01.pd.infn.it --port 8443 --queue␣
↪creamtest1 -x  /tmp/x509up_u733 --verbose
Usage: cream_serviceInfo.py [options]

cream_serviceInfo.py: error: no such option: --queue
```

In case of the errors in interacting with the CREAM CE, useful details will be provided about the failure:

```
$ ./cream_allowedSubmission.py --url https://prod-ce-01.pd.infn.it:8443 -x /tmp/
↪x509up_u733
command '/usr/bin/glite-ce-allowed-submission cream-43.pd.infn.it:8443' failed:␣
↪return_code=1
details: ['2019-12-13 15:59:57,085 FATAL - Received NULL fault; the error is due to␣
↪another cause: FaultString=[connection error] - FaultCode=[SOAP-ENV:Client] -␣
↪FaultSubCode=[SOAP-ENV:Client] - FaultDetail=[Connection refused]\n']
```

Sources are available in github

7.2.1 cream_serviceInfo.py

The serviceInfo.py retrieves information about the status of the CREAM CE. The help shows how the probe must be invoked:

```
$ ./cream_serviceInfo.py --help
Usage: cream_serviceInfo.py [options]

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -H HOSTNAME, --hostname=HOSTNAME
                        The hostname of the CREAM service.
  -p PORT, --port=PORT  The port of the service. [default: none]
  -x PROXY, --proxy=PROXY
                        The proxy path
  -t TIMEOUT, --timeout=TIMEOUT
                        Probe execution time limit. [default:
                        120 sec]
  -v, --verbose         verbose mode [default: False]
  -u URL, --url=URL     The status endpoint URL of the service. Example:
                        https://<host>[:<port>]
```

In order to get information about the CREAM service on the host https://prod-ce-01.pd.infn.it:8443, use the following command:

```
$ ./cream_serviceInfo.py --url https://prod-ce-01.pd.infn.it:8443 -x /tmp/x509up_u733
CREAM serviceInfo OK: Service Version = [1.16.4 - EMI version: 3.15.0-1.el6]
```

or similary:

```
$ ./cream_serviceInfo.py --hostname prod-ce-01.pd.infn.it --port 8443 -x /tmp/x509up_
↪u733
CREAM serviceInfo OK: Service Version = [1.16.4 - EMI version: 3.15.0-1.el6]
```

7.2.2 cream_allowedSubmission.py

This is a simple metric which checks if the submission to the selected CREAM CE is allowed. Its usage is analogous to the above metric:

```
$ ./cream_allowedSubmission.py --help
Usage: cream_allowedSubmission.py [options]

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -H HOSTNAME, --hostname=HOSTNAME
                        The hostname of the CREAM service.
  -p PORT, --port=PORT  The port of the service. [default: none]
  -x PROXY, --proxy=PROXY
                        The proxy path
  -t TIMEOUT, --timeout=TIMEOUT
                        Probe execution time limit. [default:
                        120 sec]
  -v, --verbose         verbose mode [default: False]
  -u URL, --url=URL     The status endpoint URL of the service. Example:
                        https://<host>[:<port>]
```

Notice: the use of the `--url` option is equivalent to specify both the options: `--hostname` and `--port`:

```
$ ./cream_allowedSubmission.py --hostname prod-ce-01.pd.infn.it --port 8443 -x /tmp/
↪x509up_u733
CREAM allowedSubmission OK: the job submission is ENABLED

$ ./cream_allowedSubmission.py --url https://prod-ce-01.pd.infn.it:8443 -x /tmp/
↪x509up_u733
CREAM allowedSubmission OK: the job submission is ENABLED
```

The verbose mode highlights the internal commands:

```
$ ./cream_allowedSubmission.py --url https://prod-ce-01.pd.infn.it:8443 -x /tmp/
↪x509up_u733 --verbose
executing command: /usr/bin/voms-proxy-info -timeleft
invoking allowedSubmission
executing command: /usr/bin/glite-ce-allowed-submission prod-ce-01.pd.infn.it:8443
CREAM allowedSubmission OK: the job submission is ENABLED
```

7.2.3 cream_jobSubmit.py

This metric submits a job directly to the selected CREAM CE and waits until the job termination by providing the final status. Finally the job is purged. This probe does not test the output-sandbox retrieval.

```
$ ./cream_jobSubmit.py --help
Usage: cream_jobSubmit.py [options]

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -H HOSTNAME, --hostname=HOSTNAME
                        The hostname of the CREAM service.
  -p PORT, --port=PORT  The port of the service. [default: none]
```

(continues on next page)

(continued from previous page)

```
  -x PROXY, --proxy=PROXY
                        The proxy path
  -t TIMEOUT, --timeout=TIMEOUT
                        Probe execution time limit. [default:
                        120 sec]
  -v, --verbose         verbose mode [default: False]
  -u URL, --url=URL     The status endpoint URL of the service. Example:
                        https://<host>[:<port>]/cream-<lrms>-<queue>
  -l LRMS, --lrms=LRMS  The LRMS name (e.g.: 'lsf', 'pbs' etc)
  -q QUEUE, --queue=QUEUE
                        The queue name (e.g.: 'creamtest')
  -j JDL, --jdl=JDL     The jdl path
```

The `--url` (`-u`) directive must be used to target the probe to a specific CREAM CE identified by its identifier (i.e. CREAM CE ID). Alternatively is it possible to specify the CREAM CE identifier by using the `--hostname` , `--port`, `--lrms` and `--queue` options which are mutually exclusive with respect to the `--url` option. Consider the JDL file hostname.jdl with the following content:

```
$ cat ./hostname.jdl
[
Type="Job";
JobType="Normal";
Executable = "/bin/hostname";
Arguments = "-s";
StdOutput = "std.out";
StdError = "std.err";
OutputSandbox = {"std.out","std.err"};
OutputSandboxBaseDestUri="gsiftp://localhost";
]
```

If verbose mode is disabled, the output should look like this:

```
$ ./cream_jobSubmit.py --url https://prod-ce-01.pd.infn.it:8443/cream-lsf-grid -x /
↪tmp/x509up_u733 --jdl ./hostname.jdl
CREAM JobSubmit OK [DONE-OK]
```

Notice: the use of the `--url` option is equivalent to specify both the options: `--hostname`, `--port` `--lrms` and –queue:

```
$ ./cream_jobSubmit.py --hostname prod-ce-01.pd.infn.it --port 8443 --lrms lsf --
↪queue grid -x /tmp/x509up_u733 --jdl ./hostname.jdl
CREAM JobSubmit OK [DONE-OK]
```

If the verbose mode is enabled, the output of the above command should be like this:

```
$ ./cream_jobSubmit.py --hostname prod-ce-01.pd.infn.it --port 8443 --lrms lsf --
↪queue grid -x /tmp/x509up_u733 --jdl ./hostname.jdl --verbose
executing command: /usr/bin/voms-proxy-info -timeleft
executing command: /usr/bin/glite-ce-job-submit -d -a -r prod-ce-01.pd.infn.it:8443/
↪cream-lsf-grid ./hostname.jdl
['2019-12-13 13:54:33,247 DEBUG - Using certificate proxy file [/tmp/x509up_u733]\n',
↪'2019-12-13 13:54:33,279 DEBUG - VO from certificate=[enmr.eu]\n', '2019-12-13␣
↪13:54:33,279 WARN - No configuration file suitable for loading. Using built-in␣
↪configuration\n', '2019-12-13 13:54:33,279 DEBUG - Logfile is [/tmp/glite_cream_cli_
↪logs/glite-ce-job-submit_CREAM_zangrand_20191213-135433.log]\n', '2019-12-13␣
↪13:54:33,282 INFO - certUtil::generateUniqueID() - Generated DelegationID:␣
↪[12815a52a76431b1712199d87ae5896fd6718b3a]\n', '2019-12-13 13:54:36,175 DEBUG -␣
↪Registering to [https://prod-ce-01.pd.infn.it:8443/ce-cream/services/CREAM2] JDL=[␣
↪StdOutput = "std.out"; BatchSystem = "lsf"; QueueName = "grid"; Executable = "/bin/
↪hostname"; Type = "Job"; Arguments = "-s"; JobType = "Normal";␣
↪OutputSandboxBaseDestUri = "gsiftp://localhost"; OutputSandbox = { "std.out","std.
↪err" }; StdError = "std.err" ] - JDL File=[./hostname.jdl]\n', '2019-12-13 13:54:36,
↪634 DEBUG - Will invoke JobStart for JobID [CREAM067861520]\n', 'https://prod-ce-01.
↪pd.infn.it:8443/CREAM067861520\n']
```

(continues on next page)

(continued from previous page)

```
job id: https://prod-ce-01.pd.infn.it:8443/CREAM067861520
invoking jobStatus
executing command: /usr/bin/glite-ce-job-status https://prod-ce-01.pd.infn.it:8443/
↪CREAM067861520
['\n', '******  JobID=[https://prod-ce-01.pd.infn.it:8443/CREAM067861520]\n',
↪'\tStatus        = [DONE-OK]\n', '\tExitCode      = [0]\n', '\n', '\n']
exitCode=   ExitCode      = [0]

job status: DONE-OK
invoking jobPurge
executing command: /usr/bin/glite-ce-job-purge --noint https://prod-ce-01.pd.infn.
↪it:8443/CREAM067861520
CREAM JobSubmit OK [DONE-OK]
```

7.2.4 cream_jobOutput.py

This metric extends the cream_jobSubmit.py functionality by retrieving the job's output-sandbox. Both the stage-in and stage-out phases are both performed automatically by the CE. In particular the stage-out needs the `OutputSandboxBaseDestUri="gsiftp://localhost"` set in the JDL. Finally the job is purged.

```
$ ./cream_jobOutput.py --help
Usage: cream_jobOutput.py [options]

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -H HOSTNAME, --hostname=HOSTNAME
                        The hostname of the CREAM service.
  -p PORT, --port=PORT  The port of the service. [default: none]
  -x PROXY, --proxy=PROXY
                        The proxy path
  -t TIMEOUT, --timeout=TIMEOUT
                        Probe execution time limit. [default:
                        120 sec]
  -v, --verbose         verbose mode [default: False]
  -u URL, --url=URL     The status endpoint URL of the service. Example:
                        https://<host>[:<port>]/cream-<lrms>-<queue>
  -l LRMS, --lrms=LRMS  The LRMS name (e.g.: 'lsf', 'pbs' etc)
  -q QUEUE, --queue=QUEUE
                        The queue name (e.g.: 'creamtest')
  -j JDL, --jdl=JDL     The jdl path
  -d DIR, --dir=DIR     The output sandbox path
```

The options are the same as cream_jobSubmit.py except for `--dir`. Such option allows the user to specify the path where the output-sandbox has to be stored temporarily. The default value is `/var/lib/argo-monitoring/eu.egi.CREAMCE`. Consider the JDL file hostname.jdl with the following content:

```
$ cat ./hostname.jdl
[
Type="Job";
JobType="Normal";
Executable = "/bin/hostname";
Arguments = "-s";
StdOutput = "std.out";
StdError = "std.err";
```

(continues on next page)

(continued from previous page)

```
OutputSandbox = {"std.out","std.err"};
OutputSandboxBaseDestUri="gsiftp://localhost";
]
```

If verbose mode is disabled, the output should look like this:

```
$ ./cream_jobOutput.py --hostname prod-ce-01.pd.infn.it --port 8443 --lrms lsf --
↪queue grid -x /tmp/x509up_u733 --dir /tmp --jdl ./hostname.jdl
CREAM JobOutput OK | retrieved outputSandbox: ['std.err', 'std.out']

**** std.err ****


**** std.out ****
prod-wn-038
```

Notice: the use of the `--dir` and the output-sandbox content returned in the output message.

If the verbose mode is enabled, the output of the above command should be like this:

```
$ ./cream_jobOutput.py --hostname prod-ce-01.pd.infn.it --port 8443 --lrms lsf --
↪queue grid -x /tmp/x509up_u733 --dir /tmp --jdl ./hostname.jdl --verbose
executing command: /usr/bin/voms-proxy-info -timeleft
executing command: /usr/bin/glite-ce-job-submit -d -a -r prod-ce-01.pd.infn.it:8443/
↪cream-lsf-grid ./hostname.jdl
['2019-12-13 14:02:55,478 DEBUG - Using certificate proxy file [/tmp/x509up_u733]\n',
↪'2019-12-13 14:02:55,519 DEBUG - VO from certificate=[enmr.eu]\n', '2019-12-13␣
↪14:02:55,520 WARN - No configuration file suitable for loading. Using built-in␣
↪configuration\n', '2019-12-13 14:02:55,520 DEBUG - Logfile is [/tmp/glite_cream_cli_
↪logs/glite-ce-job-submit_CREAM_zangrand_20191213-140255.log]\n', '2019-12-13␣
↪14:02:55,523 INFO - certUtil::generateUniqueID() - Generated DelegationID:␣
↪[b6b895d69f7ef0d438db82930476a2fd149d0501]\n', '2019-12-13 14:02:57,610 DEBUG -␣
↪Registering to [https://prod-ce-01.pd.infn.it:8443/ce-cream/services/CREAM2] JDL=[␣
↪StdOutput = "std.out"; BatchSystem = "lsf"; QueueName = "grid"; Executable = "/bin/
↪hostname"; Type = "Job"; Arguments = "-s"; JobType = "Normal";␣
↪OutputSandboxBaseDestUri = "gsiftp://localhost"; OutputSandbox = { "std.out","std.
↪err" }; StdError = "std.err" ] - JDL File=[./hostname.jdl]\n', '2019-12-13 14:02:58,
↪271 DEBUG - Will invoke JobStart for JobID [CREAM160637101]\n', 'https://prod-ce-01.
↪pd.infn.it:8443/CREAM160637101\n']
job id: https://prod-ce-01.pd.infn.it:8443/CREAM160637101
invoking jobStatus
executing command: /usr/bin/glite-ce-job-status https://prod-ce-01.pd.infn.it:8443/
↪CREAM160637101
['\n', '******  JobID=[https://prod-ce-01.pd.infn.it:8443/CREAM160637101]\n',
↪'\tStatus        = [IDLE]\n', '\n', '\n']
job status: IDLE
invoking jobStatus
executing command: /usr/bin/glite-ce-job-status https://prod-ce-01.pd.infn.it:8443/
↪CREAM160637101
['\n', '******  JobID=[https://prod-ce-01.pd.infn.it:8443/CREAM160637101]\n',
↪'\tStatus        = [DONE-OK]\n', '\tExitCode      = [0]\n', '\n', '\n']
exitCode=   ExitCode      = [0]

job status: DONE-OK
invoking getOutputSandbox
executing command: /usr/bin/glite-ce-job-output --noint --dir /tmp https://prod-ce-01.
↪pd.infn.it:8443/CREAM160637101
```

(continues on next page)

(continued from previous page)

```
output sandbox dir: /tmp/prod-ce-01.pd.infn.it_8443_CREAM160637101
invoking jobPurge
executing command: /usr/bin/glite-ce-job-purge --noint https://prod-ce-01.pd.infn.
↪it:8443/CREAM160637101
CREAM JobOutput OK | retrieved outputSandbox: ['std.err', 'std.out']

**** std.err ****


**** std.out ****
prod-wn-038
```

7.2.5 WN-softver probe

This probe checks the middleware version on a WN managed by the CREAM-CE. It makes use of cream_jobOutput.py in the following way:

```
$ ./cream_jobOutput.py --url https://prod-ce-01.pd.infn.it:8443/cream-lsf-grid -x /
↪tmp/x509up_u733 --dir /tmp -j ./WN-softver.jdl
CREAM JobOutput OK | retrieved outputSandbox: ['std.err', 'std.out']

**** std.err ****


**** std.out ****
prod-wn-014 has EMI 3.15.0-1.el6
```

where

```
$ cat WN-softver.jdl
[
Type="Job";
JobType="Normal";
Executable = "WN-softver.sh";
#Arguments = "a b c";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox = {"WN-softver.sh"};
OutputSandbox = {"std.out","std.err"};
OutputSandboxBaseDestUri="gsiftp://localhost";
]
```

and WN-softver.sh is attached.

The verbose option enabled gives the following output:

```
$ ./cream_jobOutput.py --url https://prod-ce-01.pd.infn.it:8443/cream-lsf-grid -x /
↪tmp/x509up_u733 --dir /tmp -j ./WN-softver.jdl --verbose
executing command: /usr/bin/voms-proxy-info -timeleft
executing command: /usr/bin/glite-ce-job-submit -d -a -r prod-ce-01.pd.infn.it:8443/
↪cream-lsf-grid ./WN-softver.jdl
['2019-12-13 14:06:25,768 DEBUG - Using certificate proxy file [/tmp/x509up_u733]\n',
↪'2019-12-13 14:06:25,804 DEBUG - VO from certificate=[enmr.eu]\n', '2019-12-13␣
↪14:06:25,805 WARN - No configuration file suitable for loading. Using built-in␣
↪configuration\n', '2019-12-13 14:06:25,805 DEBUG - Logfile is [/tmp/glite_cream_cli_
↪logs/glite-ce-job-submit_CREAM_zangrand_20191213-140625.log]\n', '2019-12-13␣
↪14:06:25,805 DEBUG - Processing file [/users/cms/zangrand/cream-nagios-master/src/
↪WN-softver.sh]...\n', '2019-12-13 14:06:25,805 DEBUG - Inserting mangled␣
↪InputSandbox in JDL: [{"/users/cms/zangrand/cream-nagios-master/src/WN-softver.sh"}
↪]...\n', '2019-12-13 14:06:25,806 INFO - certUtil::generateUniqueID() - Generated␣
↪DelegationID: [7f0ac5ec8a7deefa01f207c0b341fce1568f5282]\n', '2019-12-13 14:06:27,
↪612 DEBUG - Registering to [https://prod-ce-01.pd.infn.it:8443/ce-cream/services/
↪CREAM2] JDL=[ StdOutput = "std.out"; BatchSystem = "lsf"; QueueName = "grid";␣
```

(continues on next page)

(continued from previous page)

```
job id: https://prod-ce-01.pd.infn.it:8443/CREAM608273414
invoking jobStatus
executing command: /usr/bin/glite-ce-job-status https://prod-ce-01.pd.infn.it:8443/
↪CREAM608273414
['\n', '******  JobID=[https://prod-ce-01.pd.infn.it:8443/CREAM608273414]\n',
↪'\tStatus        = [REALLY-RUNNING]\n', '\n', '\n']
job status: REALLY-RUNNING
invoking jobStatus
executing command: /usr/bin/glite-ce-job-status https://prod-ce-01.pd.infn.it:8443/
↪CREAM608273414
['\n', '******  JobID=[https://prod-ce-01.pd.infn.it:8443/CREAM608273414]\n',
↪'\tStatus        = [DONE-OK]\n', '\tExitCode      = [0]\n', '\n', '\n']
exitCode=   ExitCode      = [0]

job status: DONE-OK
invoking getOutputSandbox
executing command: /usr/bin/glite-ce-job-output --noint --dir /tmp https://prod-ce-01.
↪pd.infn.it:8443/CREAM608273414
output sandbox dir: /tmp/prod-ce-01.pd.infn.it_8443_CREAM608273414
invoking jobPurge
executing command: /usr/bin/glite-ce-job-purge --noint https://prod-ce-01.pd.infn.
↪it:8443/CREAM608273414
CREAM JobOutput OK | retrieved outputSandbox: ['std.err', 'std.out']

**** std.err ****


**** std.out ****
prod-wn-014 has EMI 3.15.0-1.el6
```

7.2.6 WN-csh probe

This probe checks that csh is there on a WN managed by the CREAM-CE. It makes use of cream_jobOutput.py in the following way:

```
$ ./cream_jobOutput.py --url https://prod-ce-01.pd.infn.it:8443/cream-lsf-grid -x /
↪tmp/x509up_u733 --dir /tmp -j ./WN-csh.jdl
CREAM JobOutput OK | retrieved outputSandbox: ['std.err', 'std.out']

**** std.err ****


**** std.out ****
prod-wn-016 has csh
```

where

```
$ cat WN-csh.jdl
[
Type="Job";
JobType="Normal";
Executable = "WN-csh.sh";
#Arguments = "a b c";
StdOutput = "std.out";
StdError = "std.err";
```

(continues on next page)

(continued from previous page)

```
InputSandbox = {"WN-csh.sh"};
OutputSandbox = {"std.out","std.err"};
OutputSandboxBaseDestUri="gsiftp://localhost";
]
```

and WN-csh.sh is attached.

7.2.7 Deployment example

In a Nagios server version 3.5.0 testing instance, we deployed the files needed to execute the probes described above in the following directories:

```
$ ls -l /usr/libexec/argo-monitoring/probes/eu.egi.CREAMCE/
total 48
-rwxr-xr-x 1 root root  1361 Jan 30 16:58 cream_allowedSubmission.py
-rwxr-xr-x 1 root root  2972 Jan 31 12:42 cream_jobOutput.py
-rwxr-xr-x 1 root root  2972 Jan 31 12:42 cream_jobSubmit.py
-rwxr-xr-x 1 root root  1416 Jan 31 12:42 cream_serviceInfo.py
drwxr-xr-x 2 root root  4096 Jan 31 11:34 cream_cli
```

and

```
$ ls -l /etc/nagios/plugins/eu.egi.CREAMCE/
total 16
-rw-r--r-- 1 root root  213 Jan 29 14:26 hostname.jdl
-rw-r--r-- 1 root root  129 Jan 30 16:21 sleep.jdl
-rw-r--r-- 1 root root  292 Jan 31 11:34 WN-csh.jdl
-rwxr-xr-x 1 root root  603 Jan 31 11:34 WN-csh.sh
-rw-r--r-- 1 root root  300 Jan 31 11:34 WN-softver.jdl
-rwxr-xr-x 1 root root 1144 Jan 31 11:34 WN-softver.sh
```

and defined the new services adding in the file `/etc/nagios/objects/services.cfg` the following lines:

```
define service{
        use                             local-service
        host_name                       prod-ce-01.pd.infn.it
        service_description             eu.egi.CREAMCE-AllowedSubmission
        check_command                   ncg_check_native!/usr/libexec/argo-monitoring/
↪probes/eu.egi.CREAMCE/cream_allowedSubmission.py!60!-x /tmp/x509up_u733 -p 8443
        normal_check_interval           6
        retry_check_interval            3
        max_check_attempts              2
        obsess_over_service             0
}
define service{
        use                             local-service
        host_name                       prod-ce-01.pd.infn.it
        service_description             eu.egi.CREAMCE-ServiceInfo
        check_command                   ncg_check_native!/usr/libexec/argo-monitoring/
↪probes/eu.egi.CREAMCE/cream_serviceInfo.py!60!-x /tmp/x509up_u733 -p 8443
        normal_check_interval           6
        retry_check_interval            3
        max_check_attempts              2
        obsess_over_service             0
}
```

(continues on next page)

(continued from previous page)

```
define service{
        use                             local-service
        host_name                       prod-ce-01.pd.infn.it
        service_description             eu.egi.CREAMCE-JobSubmit
        check_command                   ncg_check_native!/usr/libexec/argo-monitoring/
↪probes/eu.egi.CREAMCE/cream_jobSubmit.py!60!-x /tmp/x509up_u733 -p 8443 -l lsf
-q creamtest1 -j /etc/nagios/plugins/eu.egi.CREAMCE/hostname.jdl
        normal_check_interval           6
        retry_check_interval            3
        max_check_attempts              2
        obsess_over_service             0
}
define service{
        use                             local-service
        host_name                       prod-ce-01.pd.infn.it
        service_description             eu.egi.CREAMCE.WN-Softver
        check_command                   ncg_check_native!/usr/libexec/argo-monitoring/
↪probes/eu.egi.CREAMCE/cream_jobOutput.py!60!-x /tmp/x509up_u733 -p 8443 -l lsf
-q creamtest1 -j /etc/nagios/plugins/eu.egi.CREAMCE/WN-softver.jdl
        normal_check_interval           6
        retry_check_interval            3
        max_check_attempts              2
        obsess_over_service             0
}
define service{
        use                             local-service
        host_name                       prod-ce-01.pd.infn.it
        service_description             eu.egi.CREAMCE.WN-Csh
        check_command                   ncg_check_native!/usr/libexec/argo-monitoring/
↪probes/eu.egi.CREAMCE/cream_jobOutput.py!60!-x /tmp/x509up_u733 -p 8443 -l lsf
-q creamtest1 -j /etc/nagios/plugins/eu.egi.CREAMCE/WN-csh.jdl
        normal_check_interval           6
        retry_check_interval            3
        max_check_attempts              2
        obsess_over_service             0
}
```

The check_command ncg_check_native was defined in the file `/etc/nagios/objects/commands.cfg` as below:

```
define command{
        command_name                    ncg_check_native
        command_line                    $ARG1$ -H $HOSTNAME$ -t $ARG2$ $ARG3$
}
```

CHAPTER 8

Releases

8.1 Release 1.16.7

The current release of the CREAM service is fully compatible with the latest version of Bouncycastle, CAnL and VOMS published in EPEL.

The fixed issues are:

- GGUS-Ticket-ID: #106384
- GGUS-Ticket-ID: #122974
- GGUS-Ticket-ID: #124404
- GGUS-Ticket-ID: #127020
- GGUS-Ticket-ID: #133522
- GGUS-Ticket-ID: #136074

The list of packages for the CREAM services on CentOS 7 is the following:

Package	Version	Arch
BLAH	1.22.3-1	x86_64
BLAH-debuginfo	1.22.3-1	x86_64
glite-ce-common-java	1.16.5-2	noarch
glite-ce-common-java-doc	1.16.5-2	noarch
glite-ce-cream	1.16.5-5	noarch
glite-ce-cream-api-java	1.16.6-3	noarch
glite-ce-cream-core	1.16.5-5	noarch
glite-ce-cream-utils	1.3.7-1	x86_64
info-dynamic-scheduler-slurm	1.0.5-1	noarch

The list of packages for the CREAM services on Scientific Linux 6 is the following:

Package	Version	Arch
canl-java-tomcat	0.1.19-2	noarch
glite-ce-blahp	1.22.3-1	x86_64
glite-ce-blahp-debuginfo	1.22.3-1	x86_64
glite-ce-common-java	1.16.5-2	noarch
glite-ce-common-java-doc	1.16.5-2	noarch
glite-ce-cream	1.16.5-5	noarch
glite-ce-cream-api-java	1.16.6-3	noarch
glite-ce-cream-core	1.16.5-5	noarch
glite-ce-cream-utils	1.3.7-1	x86_64
info-dynamic-scheduler-lsf	2.3.8-1	noarch
info-dynamic-scheduler-lsf-btools	2.3.8-1	noarch
info-dynamic-scheduler-slurm	1.0.5-1	noarch
lcg-info-dynamic-scheduler-condor	1.1-1	noarch
lcg-info-dynamic-scheduler-pbs	2.4.6-1	noarch

The basic requirement for the installation is UMD 4.7.1. The java application requires Java SE 8 (OpenJDK 1.8.0 or later). The installation and configuration of the CREAM CE site is certified with the puppet module version 0.1.1 or greater. YAIM is no more supported on Scientific Linux 6 and its usage must be considered at "your own risk".

The update process on Scientific Linux 6 requires the following additional steps:

- After the update it is necessary to remove the packages *bouncycastle-mail* and *bouncycastle* (version 1.46).
- Any broken link in */var/lib/tomcat6/webapps/ce-cream/WEB-INF/lib* and */usr/share/tomcat6/lib* must be manually deleted.

If YAIM is used for the configuration it is necessary to create a symbolic link from */usr/share/java/bcprov-1.58.jar* into */usr/share/java/bcprov.jar* and re-running the configurator.

8.2 Release 1.16.6

The changes delivered with this release are:

- Support for CentOS 7
- Support for accelerator devices (NVidia GPUs, MIC)
- Implementation of GLUE 2.1 v.0.4 (Draft)

The list of packages for the CREAM services is the following:

Package	Version	Arch
BLAH	1.22.2-1	x86_64
BLAH-debuginfo	1.22.2-1	x86_64
axis2	1.6.4-2.emi	noarch
canl-java-tomcat	0.2.1-1	noarch
condor-classads-blah-patch	0.0.1-1	x86_64
dynsched-generic	2.5.5-2	noarch
glite-ce-common-java	1.16.5-1	noarch
glite-ce-common-java-doc	1.16.5-1	noarch
glite-ce-cream	1.16.5-3	noarch
glite-ce-cream-api-java	1.16.6-2	noarch
glite-ce-cream-core	1.16.5-3	noarch
glite-ce-cream-utils	1.3.6-2	x86_64
glite-ce-wsdl	1.15.1-1	noarch
glite-info-dynamic-ge	7.2.0-29.1	noarch
glite-jdl-api-java	3.3.2-3	noarch
glite-jdl-api-java-doc	3.3.2-3	noarch
info-dynamic-scheduler-lsf	2.3.8-3	noarch
info-dynamic-scheduler-lsf-btools	2.3.8-3	noarch
info-dynamic-scheduler-slurm	1.0.4-1	noarch
jclassads	2.4.0-3	noarch
kill-stale-ftp	1.0.1-1	noarch
lcg-info-dynamic-scheduler-condor	1.1-1	noarch
lcg-info-dynamic-scheduler-pbs	2.4.6-1	noarch

The list of packages for the CREAM clients is the following:

Package	Version	Arch
glite-ce-cream-cli	1.15.4-1	x86_64
glite-ce-cream-client-api-c	1.15.5-1	x86_64
glite-ce-cream-client-devel	1.15.5-1	x86_64
glite-jdl-api-cpp	3.4.4-1	x86_64
glite-jdl-api-cpp-devel	3.4.4-1	x86_64
glite-jdl-api-cpp-doc	3.4.4-1	x86_64
glite-wms-utils-classad	3.4.3-1	x86_64
glite-wms-utils-classad-devel	3.4.3-1	x86_64
glite-wms-utils-exception	3.4.3-1	x86_64
glite-wms-utils-exception-devel	3.4.3-1	x86_64
log4cpp	1.0.13-1	x86_64
log4cpp-devel	1.0.13-1	x86_64

The installation and configuration of the CREAM CE site is certified with the puppet module version 0.0.16 or greater.

The supported batch systems are:

Batch system	Version
TORQUE	4.2.10
SLURM	16.05.10
Htcondor	8.6.3
LSF	7.0
GridEngine	6.2

The fixed issues are:

- GGUS-Ticket-ID: #106384
- GGUS-Ticket-ID: #122974
- GGUS-Ticket-ID: #124034
- GGUS-Ticket-ID: #124404
- GGUS-Ticket-ID: #127020

The known issues are:

- The CREAM UI requires classads libraries up to version 8.4.11, it does not work with versions 8.6.*
- GridEngine is partially supported, the infoprovider does not publish informationa about acceleratore devices.
- The puppet agent may report parsing errors.
- If HTCondor is the batch system adopted, the HTCondor services on the computing element must be restarted after the installation of the CREAM service.

CHAPTER 9

Indices and tables

- genindex
- modindex