NORDUGRID-MANUAL-20
3/4/2012

ARC Computing Element

System Administrator Guide

F. Paganelli, Zs. Nagy, O. Smirnova,
and various contributions from all ARC developers

Contents

Chapter 1

Overview

The *ARC middleware* [**?**] by *NorduGrid* [**?**] is a software solution that uses grid technologies to enable *sharing* and *federation* of *computing* and *storage* resources distributed across different administrative and application domains. ARC is used to create grid infrastructures of various scope and complexity, from campus to national grids.

This document gives a detailed overview of the ARC *Computing Element* (CE), along with step-by-step installation and configuration instructions and a full reference of the configuration commands.

1.1 The grid

ARC-based grid aggregates computing and storage resources, making them accessible through standard interfaces, and using a common information system to optimize access.

Client tools can query this information system to see what kind of resources are available, match user's tasks to best available resources, submit computing jobs, which are smaller or bigger tasks (scripts and/or binaries, often processing defined input data) to run on computing nodes in the grid, they can access files on and upload results to storage resources.

For users, all this complexity is hidden: they simply formulate their tasks in a special language and send them to the grid, not even knowing which computing or storage resources are out there. ARC takes care of the rest.

While submitting jobs, users must specify requirements for each job, namely, what software should it execute, what data to process, what kind of software environment it needs on the computing node, how much memory, how strong CPU etc—these are specified in the formal job description. They can use various client tools, like the native command-line interface supplied along with the ARC middleware [**?**], GUI tools, web portals or specialized clients as part of a bigger software tool. All users must be authenticated by grid services using *X.509 certificate*s signed by trusted Certificate Authorities. ARC also uses short-lived *proxy certificates* to delegate users' rights to various activities performed by Grid services on their behalf, such as job execution or data transfer. Authentication alone is not sufficient: users must also be authorized to perform such activities. Typically, users form groups (called *Virtual Organization*s, VOs) to ease to process of getting authorized on the several computing resources.

In order to handle all the computing resources in a uniform way, there is a need for a layer ("middleware") between the client tools and the resources: the *Computing Element* (CE). This document describes how to use the CE functionality of the ARC middleware to make a computing resource accessible for grid users.

1.2 The ARC services

Grid computing has three big areas: *computation*, *storage* and *information*. The server side of the ARC middleware provides services for all three main areas:

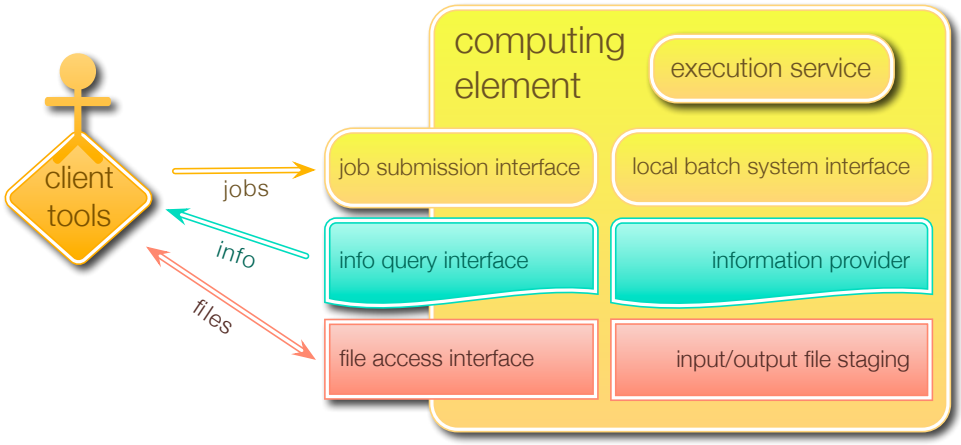


Figure 1.1: The interfaces and internal components of a generic grid computing element

- The **Computing Element (CE)**. By installing the ARC *Computing Element* (CE), a computing resource (usually, computing clusters managed by a batch system—*LRMS*—or a standalone workstation) will gain standard grid interfaces, through which users (authenticated using their *X.509 certificates*) can get information about the resource, submit, query and manage computing jobs with the help of client tools. The computing resource will also gain a capability to register itself to several different grid information system such that client tools would discover it.
- The **Storage Element (SE)**. The ARC *GridFTP Server* [**?**] besides being an important part of the ARC Computing Element, can also be installed as a standalone storage solution.
- The **Indexing Service (EGIIS)**. The ARC *Enhanced Grid Information Indexing Service* (EGIIS) is capable of collecting registrations from computing elements and storage elements equipped with ARC Resource Information Service (ARIS) and providing these resource pointers to the client tools. There are several EGIIS instances deployed all around the world. New resources usually register themselves to one or more of the existing indexes.

These three functionalities are implemented by one or more ARC services, which can be installed separately in a standalone manner, or all of them can reside on the same machine. *This document only describes the ARC Computing Element (CE).* For the description of the standalone GridFTP Storage Element, please refer to the The NorduGrid GridFTP Server document [**?**].

There is a very important forth area: the client side. The **ARC command line clients** [**?**] are able to fully interact with the A-REX or other computing elements, they support several data transfer protocols to be able to upload and download files from all kinds of storage resources. They are querying the available computing resources from the information system, doing brokering based on the requirements specified in the job description (languages supported: XRSL [**?**], JSDL [**?**] and JDL [**?**]), they are able to query the status of jobs and manage their lifecycle, and to handle all aspects of the secure communication including delegation of the user's credentials.

1.3 The functionality of the ARC Computing Element

Figure 1.1 shows the interfaces and the internal components of a generic grid computing element. An ARC **Computing Element (CE)** has these interfaces and components, and with them it is capable of the following:

- to advertise (register) itself in an information system to make the clients tools know about its location and capabilities
- to accept job execution requests coming through the *job submission interface* and to process the jobs (written in standard job description languages) handled by the *execution service*

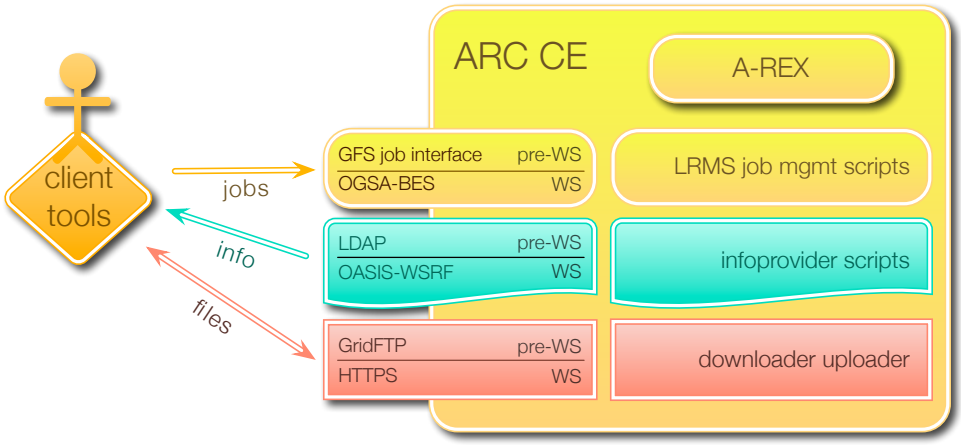


Figure 1.2: The interfaces and components of the ARC Computing Element

- to accept the files requested by the jobs from the user through the *file access interface* or to download them from remote storages (*input file staging*) and to avoid downloading the same files over and over again by caching them
- to forward the jobs to the *local resource management system (LRMS)* (such as Condor [**?**], Torque [**?**], OpenPBS [**?**], Sun Grid Engine [**?**], etc.), which will schedule and execute them on the computing nodes in the local cluster
- to monitor the status of the jobs by running the *information provider* scripts and make this information available through the *information query interface*.
- to make the results (output files) of the jobs accessible through the *file access interface* or upload them to a remote storage *output file staging*

1.4 The A-REX, the execution service

The most important component of the ARC Computing Element is the **A-REX** (ARC Resource-coupled EXecution service). The A-REX accepts requests containing a description of generic computational jobs and executing it in the underlying local batch system. It takes care of the pre- and post-processing of the jobs: *staging in* (downloading) files containing input data or program modules from a wide range of sources and storing or *staging out* (uploading) the output results.

The ARC Computing Element with the help of A-REX and some other services provides two distinct set of interfaces: the pre-web service interfaces, which are based on LDAP and GridFTP, and are currently widely deployed and in production; and the web service interfaces, which are based on grid standards, are also well-tested and production-quality but not yet widely used. Figure 1.2 shows the interfaces and also the other components.

1.4.1 The pre-web service interfaces

The pre-web service *job submission interface* uses the GridFTP protocol in a special way. It is provided by a separate component, the ARC *GridFTP Server* (GFS) has a *job plugin* which accepts job descriptions in the *XRSL* job description language. The A-REX works together with the GridFTP Server to get notified about new jobs.

The pre-web service *information query interface* of the ARC CE is an LDAP/BDII based interface, which is provided by a separate component, called the *ARIS* (the ARC Resource Information System).

The pre-web service *file access interface* uses the GridFTP protocal, and is served by the same ARC *GridFTP Server* (GFS) which provides the job submission interface too.

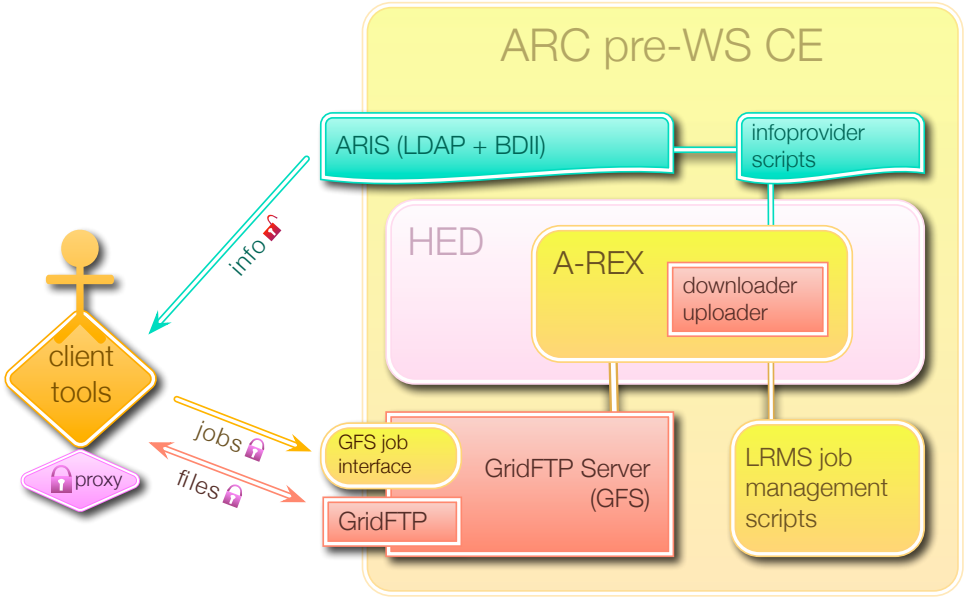


Figure 1.3: The services and components of the pre-web service ARC CE

The A-REX service itself has no direct interface to the clients in the pre-web service case, it communicates through the GridFTP Server (GFS). Figure 1.3 shows the services and the components of the pre-web service ARC CE.

1.4.2 The web service interfaces

The web service *job submission interface* of the ARC CE is provided by the A-REX itself, and it is a standard-based interface: an enhancement of the OGSA Basic Execution Service recommendation [**?**]).

The web service *information query interface* of the ARC CE is also provided by the A-REX itself, and it is also a standard-based interface, called *LIDI* (Local Information Description Interface), which is an implementation of the OASIS Web Services Resource Properties specification [**?**].

The *file access interface* is technically not a web service, but it is the well-known HTTPS interface provided by the A-REX itself.

In the web service case, all the interfaces are provided by the A-REX itself, there is no need of separate services. Figure 1.4 shows the components of the web service ARC CE.

The web service and the pre-web service interfaces are capable to work together: an ARC CE can provide both interfaces at the same time.

1.5 Security on the Grid

Security on the grid is achieved using *X.509 certificates*. Any grid service needs to have a certificate issued by a trusted *Certificate Authority (CA)*. A single machine, like a front-end running a CE, is identified by a **host certificate**. A single user accessing the grid is identified by a **user certificate** also issued by a trusted CA.

Grid CAs are often established in each country, though there are also CAs issuing certificates for specific organizations (like CERN), or for several countries (like TERENA). Each CA has its own certification policies and procedures: to access/setup a grid service, one has to contact the relevant Certificate Authority in order to obtain the needed certificates.

When a user wants to access the grid, the client tools generate a short-lived *proxy certificate* to delegate user's rights to jobs or other activities performed by grid services on user's behalf.

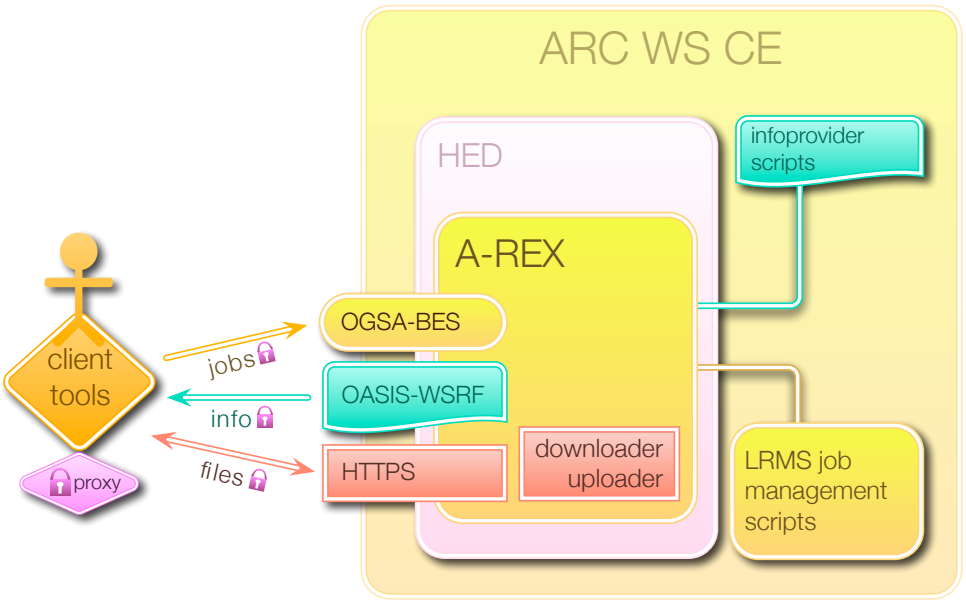


Figure 1.4: The components of the web service ARC CE

In order for the server to authenticate the client, the certificate of the CA issuing the user's certificate has to be installed on the server machine. In the same manner in order for the client to authenticate the server, the certificate of the CA issuing the host's certificate should be installed on the client machine.

On the server side it is the responsibility of the system administrator to decide which authorities to trust, by installing each authority's certificate. On the client side, the user decides which CA certificates she installs. The user cannot access a grid resource, if the issuer CA certificate of the host is not installed.

Figure 1.5 shows an overview of the required keys and certificates, and also the process of creating a client proxy certificate using the user's credentials, and optionally collecting more information about the Virtual Organization (VO) the user belongs by connecting to a Virtual Organization Membership Service (VOMS).

1.6 Handling jobs

A job is described as a set of input files (which may include executables), a main executable and a set of output files. The job's life cycle (its session) starts with the arrival of the job description to the Computing Element (CE), next comes the gathering of the input files, then follows the execution of the job, then the handling of the output files and finally job ends with the removal of the session contents by either the user or after a specified amount of days by the CE.

Each job gets a directory on the CE called the *session directory* (SD). Input files are gathered in the SD. The job may also produce new data files in the SD. The A-REX does not guarantee the availability of any other places accessible by the job other than SD (unless such a place is part of a requested *Runtime Environment*, see section X.Y.).

Each job gets a globally unique identifier (*jobid*). This *jobid* is effectively a URL, and can be used to access the session directory (to list, download and even upload files into the SD) from outside, either through the HTTP(S) interface or through the GridFTP Server.

1.6.1 A sample job processing flow

The jobs in the ARC Computing Element usually go through these steps:

1. The client (such as the ARC command line tools [**?**]) connects to the *job submission interface* (either to the web service interface of A-REX or to the GridFTP Server).

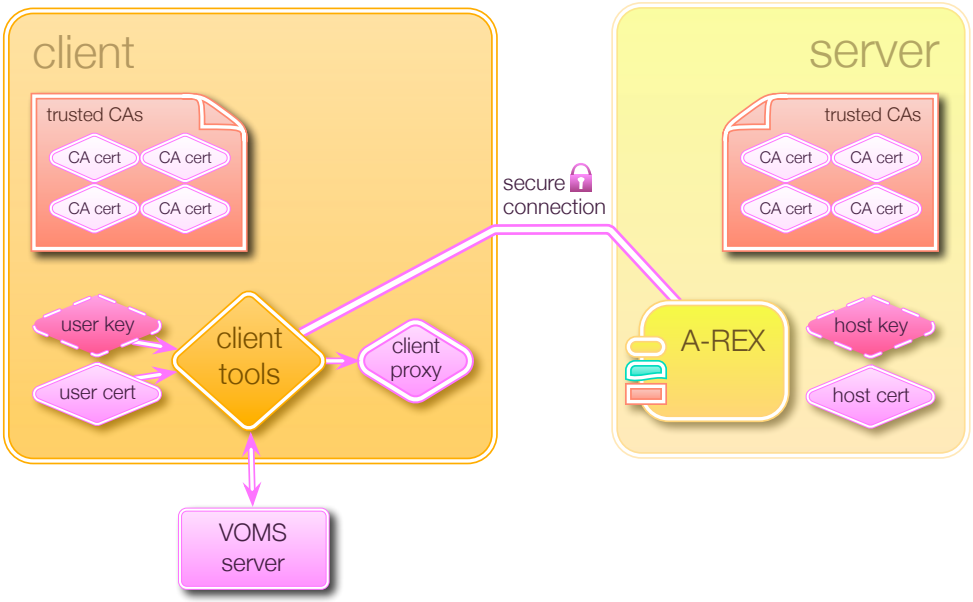


Figure 1.5: Certificates on the client side and on the server side. The client tools create a proxy certificate using the user's credentials, and optionally collect more information about the Virtual Organization (VO) the user belongs by connecting to a Virtual Organization Membership Service (VOMS).

2. Using the well-established processes of the X.509 Public-Key Infrastructure [**?**], the client and the server both authenticate each other, based on the trusted CA credentials which were previously installed on both ends.

3. The A-REX authorizes the user based on configurable rules, and maps the grid identity to a local username which should be available also on all the worker nodes.

4. The client tool delegates user's credentials to the A-REX to enable it to act on behalf of the user when transferring files. (See Figure 1.6.)

5. A job description written in one of the supported languages (XRSL [**?**] or JSDL [**?**]) is sent from the client to the server. (The client itself understands the JDL [**?**] language also, and it translates it to either XRSL or JSDL for the A-REX to understand.)

6. The job is accepted and a directory (the *session directory, SD*) is created which will be the home of the session. Metadata about the job is written into the *control directory* of the A-REX.

7. The client tool receives the location of the session directory (SD), and if there are local input files, those will be uploaded into the SD through the *file access interface* (either through the HTTP(S) interface of the A-REX, or through the GridFTP Server).

8. If the job description specifies input files on remote locations, the A-REX fetches the needed files and puts them into the SD. If the caching is enabled, the A-REX checks first if the file was already downloaded recently, and uses the cached version if possible.

9. When all the files prescribed in the job description are present (either uploaded by the client tool or downloaded by the A-REX), a suitable job script is created for and submitted to the configured batch system (LRMS).

10. During this time, the SD of the job is continuously accessible by the client tool, thus any intermediate result can be checked.

11. The *information provider* scripts periodically monitor the job status, updating the information in the control directory.

12. When the job in the LRMS is finished, the A-REX uploads, keeps or removes the resulted output files according to the job description.

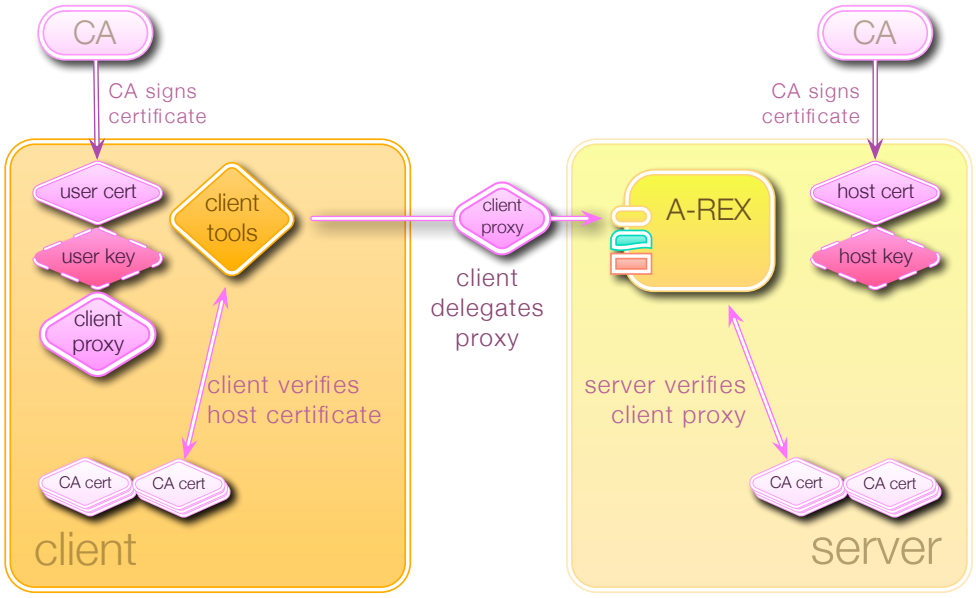


Figure 1.6: The client delegates the client proxy to the Computing Element, while both parties verifies that the credentials are signed by a trusted Certificate Authority (CA)

13. The client tool may also download the output files through the *file access interface*, and remove the job from the Computing Element (CE).

During the whole lifetime of the job, its status can be queried through the *information query interface* (either through the LDAP interface or through the LIDI web service interface).

Figure 1.7 and Figure 1.8 shows the staging process.

1.7 The runtime environments

The following text was taken from the following site: `http://gridrer.csc.fi/intro.phtml`.

Grid Runtime Environments (REs) provide user interfaces to application software (and some other) resources in a way that is independent of the details of the local installation of the application and computing platform (OS, hardware, etc.). This page introduces Runtime Environments as regarded by NorduGrid ARC middleware.

Runtime environments are typically required by large research groups or user bases, dealing with a common set of software. Whether a particular application should be implemented as a RE or not, depends on the existence of large enough user base, RE Homepage maintainer and site administrators able/willing to install the RE.

The actual implementation of particular RE may differ from site to site as necessary. However, it should be designed so that resource providers with different accounting, licence or other site-specific implementation details can advertise the same application interface (RE) for all users. It is always up to the local system administrators to take a decision whether to install and enable a particular runtime environment or not.

We hope that using REs will provide an additional benefit for site administrators by reducing work of maintaining applications.

Runtime Environment is composed of two parts:

1. RE Homepage
   - describes the users' application interface
   - provides application installation instructions for the site administrators

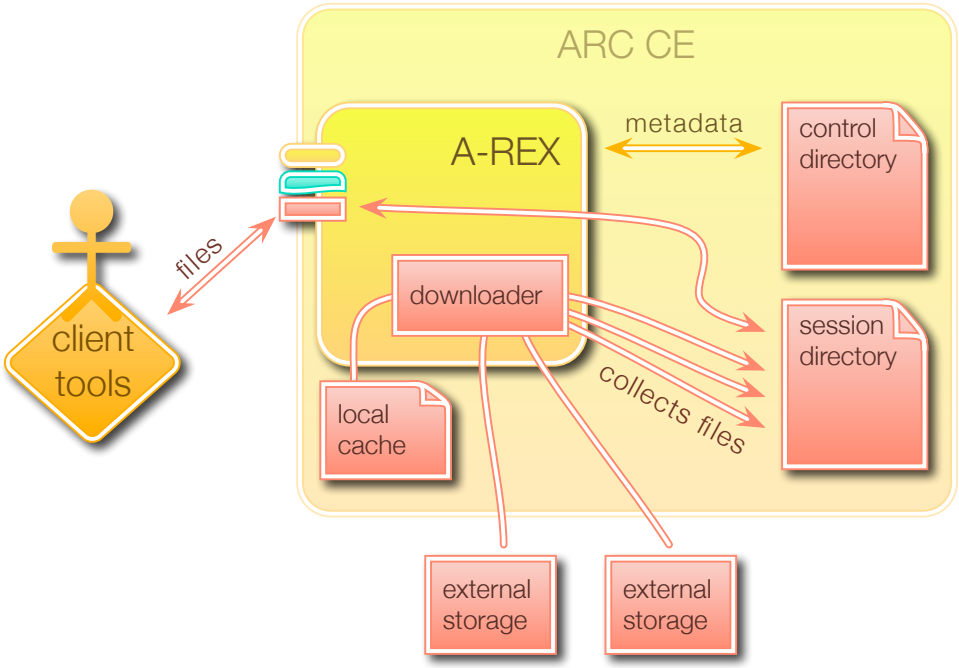


Figure 1.7: The process of staging in the input files of a job

- links to the application support information

2. RE itself
   - is a shell environment initialization script
   - is installed on computing resources
   - initializes variables that point to the application software

Let's have an example from the user perspective:

A user has a script written in python 2.6 that she wishes to execute in some remote computing node in Grid. She requests `PYTHON-2.6` Runtime Environment in the job-description file and passes that file to the her local Grid User Interface (arcsub), i.e. submits the job.

After the submission arcsub parses the job description, notices the RE request and submits the job only to clusters advertising that RE. The remote cluster's grid-manager initializes the environment in the computing node before the execution. It initializes the environment so that python interpreter and standard libraries are in the PATH and executable/readable by the user as described in the RE Homepage.

What does this give to the users:

- easier access to a large software resource base
- identical interface to applications independent of the computing platform

What does this do for resource providers and application developers:

- opens the application to a large user base
- reduces overlapping work with application support

Please follow this URL for more information: `http://gridrer.csc.fi/admins.phtml`

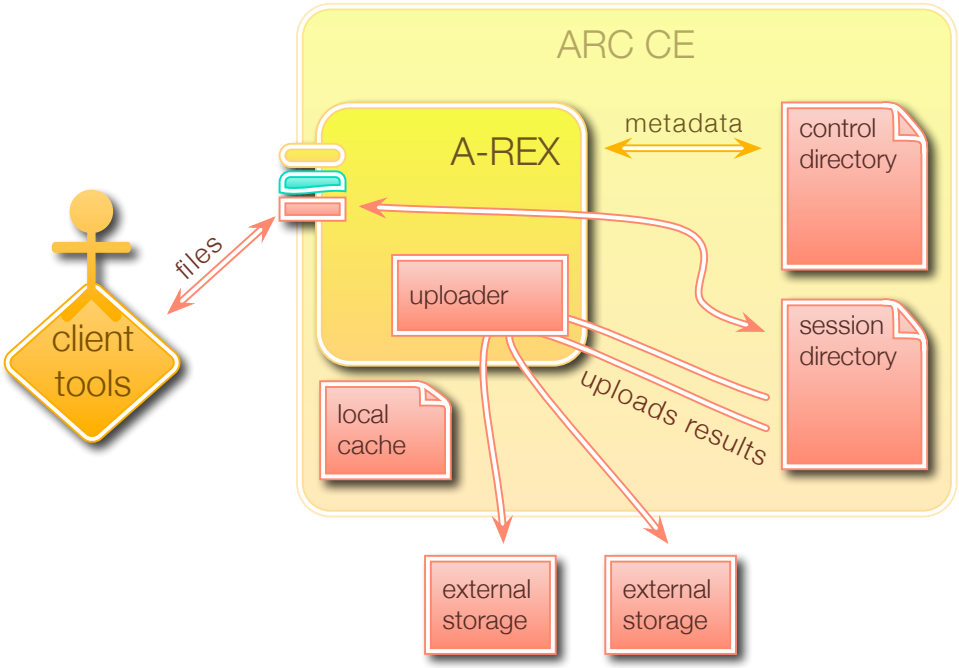


Figure 1.8: The process of staging out the output files of a job

1.8 The local information

In order to create a Grid infrastructure using ARC-enabled computing resources, information description and aggregation services need to be deployed. ARIS is coupled to a computing resource and collects information about it. EGIIS keeps a list of ARIS instances, and eventually, of other EGIIS instances lower down in hierarchy. Top-level EGIIS instances thus serve as an entry point to the Grid, allowing to discover all the resources.

While ARIS is *coupled* to a resource, EGIIS is an *independent* service. A typical Grid resource owner always has to deploy ARIS*. EGIIS servers, on the other hand, are normally deployed by the overall Grid infrastructure operators.

A system effectively created by ARIS and EGIIS services is called the *ARC Information System*. Being based on OpenLDAP [**?**], it can be accessed in a standard manner by a variety of LDAP clients, giving a full overview of the infrastructure resources.

ARIS instances are responsible for resource (e.g. computing or storage) description and characterization. The local information is generated on the resource, and it can be cached. Upon client requests it is presented via LDAP interface.

1.8.1 Overview of ARC LDAP Infosys schemas

ARC information system currently can present information in three different formats, or schemas. These can be enabled simultaneously. The schemas are:

1. NorduGrid-ARC schema – this is the NorduGrid default schema, described in detail in this document. It was inspired by Globus MDS, but has been improved a lot over the years and due to incompatible changes was moved into the NorduGrid LDAP namespace. If you want standard NorduGrid clients to submit jobs to your resource, you want to publish this schema.

2. Glue 1.2 – This is the schema that is used by gLite [**?**]. Currently, gLite support Glue 1.3 schema, but Glue 1.2 is sufficient to be compatible. If you configure ARC to publish information in the Glue 1.2

*Without ARIS, a resource is still functional, but is not a Grid resource

format, you will first produce data in NorduGrid-ARC schema which will then be translated to Glue 1.2. If you want to allow gLite clients to submit to your resource, you want to publish this schema. Please note, that you will also need to hook in your ARC cluster into the gLite information system in order to get this interoperability to work.

3. Glue 2.0 – This is the schema that will become the common schema for the EMI [**?**]. This schema can be published both through LDAP and XML interfaces of ARC Compute Element.

ARIS is the information service that is installed on the ARC Compute Element. It publishes via LDAP interface information about the local computing cluster, like: operating system, amount of main memory, computer architecture, information about running and finished jobs, users allowed to run and trusted certificate authorities. The information can be published in either NorduGrid-ARC schema, Glue 1.2 schema or Glue 2.0 schema.

The dynamic resource state information is generated on the resource. Small and efficient programs, called information providers, are used to collect local state information from the batch system, from the local Grid layer (e.g. A-REX, Grid Manager or GridFTP server) or from the local operating system (e.g. information available in the `/proc` area). Currently, ARC is capable interfacing to the following batch systems (or local resource management system LRMS in the ARC terminology): UNIX fork, the PBS-family (OpenPBS, PBS-Pro, Torque), Condor, Sun Grid Engine, IBM LoadLeveler and SLURM.

The output of the information providers (generated in LDIF format) is used to populate the local LDAP tree. This OpenLDAP back-end implements two things: it is capable caching the providers output and upon client query request it triggers the information providers unless the data is already available in its cache. The caching feature of the OpenLDAP back-end provides protection against overloading the local resource by continuously triggering the information providers.

1.9 LRMS, Queues and execution targets

Usually the A-REX is installed on top of an existing local resource management system (LRMS). The A-REX has to interfaced to the LRMS in order to be able to submit jobs and query their information. Currently, ARC is capable interfacing to the following batch systems: UNIX fork, the PBS-family (OpenPBS, PBS-Pro, Torque), Condor, LFS, Sun Grid Engine, IBM LoadLeveler and SLURM.

The A-REX assumes that the LRMS has one or more **queue**s, which is a couple of (usually homogeneous) worker nodes grouped together. These queues should not overlap. The different LRMSes has different concept of queues (or has no queues at all). Nevertheless, in the A-REX configuration, the machines of the LRMS should be mapped to A-REX queues. The details can be found in Section **4.4.2**, *Connecting to the LRMS*.

The client side job submission tools query the information system for possible places to submit the jobs, where each queue on a CE represented as an **execution target**, and treated separately.

Chapter 2

Requirements

To properly configure an ARC CE the following prerequisites are needed:

- **Administrators installing ARC CE must have access to network firewall configuration**: Several ports will need to be open for the ARC services to work (see **4**, *Configuration* and **4.1.4**, *Firewalls*
- **Time Syncronization** of the system that will run an ARC CE must be setup, by using the NTP protocol [] or similar. The grid relies on syncronization for the jobs to be correctly submitted and for the security infrastructure to work properly.

The following is optional but suggested to be on the machines running an ARC CE:

- **A networked filesystem** such as NFS or similar, to connect storage and share job data between the ARC middleware and the LRMS system behind it.

2.1 Software Requirements

ARC services can be built mainly for GNU/Linux and Unix systems.

Table 2.1 shows the current officially supported ones.

Operating System	Version/Distribution	Supported Architectures
GNU/Linux	Scientific Linux 5.5+	i386, x86_64
	RedHat 5+	i386, x86_64
	Debian 6+	i386, x86_64
	Ubuntu 10.04+	i386, x86_64

Table 2.1: Supported operating systems

For a detailed list of the software libraries needed to compile and install ARC services, please refer to the README included in the source tarball. See Chapter **3**, *Installation* for details.

2.2 Hardware Requirements

The NorduGrid middleware does not impose heavy requirements on hardware. The choice is only bound to the computational needs of your organization.

Table 2.2 shows the minumum requirements.

Architecture	32 or 64 bits
CPU families	$\geq$ i386 , PowerPC
CPU Speed	$\geq$ 300 MHz
Memory Size	$\geq$ 128MB
Disk space for binaries	$\leq$ 30MB
Disk space including development files	160MB
Disk space including external software (such as Globus Toolkit 5)	+10MB
Network connectivity	a public IP on the front-end cluster is strongly encouraged. Worker nodes can be on a private or local network.

Table 2.2: Hardware Requirements

2.3 Certificates

To run an ARC CE and have it servicing the grid, a **host certificate** provided by a Certificate Authority (CA) is needed.

A request for such a certificate must be sent to the National Grid Infrastructure organization or to any local organization entitled to provide grid services.

The CA certificate is needed as well, this is public and can be usually obtained from either the CA itself, of fetched from EMI repository, IGTF repository, NorduGrid yum/apt repositories, or from the NorduGrid Downloads area. These are needed to verify that the service and the users connecting to it have valid credentials, to perform mutual authentication.

If this is the **first time** the reader sets up an ARC CE, we suggest to obtain temporary test certificates for hosts, users and a temporary CA via the InstantCA service:

`https://arc-emi.grid.upjs.sk/instantCA/instantCA`

Such certificates cannot be used in production environments and can only be used for testing purposes.
Once the system administrator feels comfortable with an ARC CE setup, InstantCA certificates can be substituted with actual ones from trusted production CAs.

Installation of certificates is discussed in Section **3.3**, *Installation of certificates*.

Chapter 3

Installation

3.1 Installation for commom GNU/Linux Distributions

The preferred installation method for ARC middleware is by installing packages from **repositories**. The currently supprted distributions are those based on **YUM-RPM** (Redhat, CentOS, Fedora, Scientific Linux) and those based on **APT** (Debian, Ubuntu).

The packaging systems will automatically download additional libraries and dependencies for all the ARC middleware components to work properly. You can choose to install single packages one by one and add functionalities in a step-by-step fashion. Please refer to table 3.1 if you plan to do so.

ARC provides also meta-packages that are shortcuts to install a group of packages that provide a single functionality. It is strongly recommended to use this functionality for a quick start.

3.1.1 Setting up the repositories

The current repository is the official Nordugrid one. To configure nordugrid repositories please follow the up-to-date instructions at:

`http://download.nordugrid.org/repos.html`

If ARC CE is to be used together with other european grid products, for example to join european scientific experiments such as ATLAS or ALICE, then the suggested repository is the EMI repository.

The EMI consortia provides also official production level customer support for distributions such as Scientific Linux 5.5 and Debian 6 and above, so it is strongly recommended to install from EMI if you are planning to use and ARC CE on these systems.

To install such repositories, please follow the instructions at EMI official website at this link:

`http://emisoft.web.cern.ch/emisoft/index.html`

3.1.2 Performing the installation

To perform the installation, follow these steps:

1. Configure a repository (see above for details)
2. Install the ARC CE using meta-packages: issue the following command as root:
   For RPM-Based distros:

```
yum install nordugrid-arc-compute-element
```

For APT-Based distros:

```
apt-get install nordugrid-arc-compute-element
```

This will install the packages marked with * in table 3.1.

3. (optional) if you want to customize your setup with individual packages, issue:
   For RPM-Based distros:

```
yum install <packagename>
```

For APT-Based distros:

```
apt-get install <packagename>
```

3.2 Installation for other systems and distributions

Packages are not provided for platforms other than GNU/Linux, so for the moment being the only way of installing ARC services is by compiling from source. Please refer to the README file* in the source code repository for more details.

3.3 Installation of certificates

A description of what certificates are and why are needed can be found in Section **1.5**, *Security on the Grid*.

Information about reading the contents of the certificates, changing their formats and more can be found in the ARC certificate mini how-to document†.

In case ARC was installed using packages (see Chapter **3**, *Installation*) all the required CAs are already installed and a script will automatically update them together with system updates.

If you want to install or remove specific CAs, NorduGrid repositories contain packaged CAs for ease of installation. By installing these packages, all the CA credentials will get updated by system updates. These packages are named in this format:

```
ca_<CA name>
```

Example:

```
ca_nordugrid
```

You can install them as you would install any package by APT or YUM.

In case your resource is in a Nordic country (Denmark, Finland, Norway, Iceland or Sweden), install the `certrequest-config` package from the NorduGrid Downloads area. It is also in the nordugrid repositories with name `ca-nordugrid-certrequest-config`. This contains the default configuration for generating certificate requests for Nordic-based services and users. If you are located elsewhere, contact your local CA for details.

For example, in Nordic countries, generate a host certificate request with

```
grid-cert-request -host <my.host.fqdn>
```

*http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/README
†http://www.nordugrid.org/documents/certificate_howto.html

Package	Content
All	
nordugrid-arc*!	All components
ARC CE	
nordugrid-arc-arex*!	ARC Remote EXecution service
nordugrid-arc-hed*!	ARC Hosting Environment Daemon
nordugrid-arc-plugins-needed*!	ARC base plugins
nordugrid-arc-gridftpd*!	ARC gridftp server
nordugrid-arc-plugins-globus*	ARC Globus plugins
nordugrid-arc-python	ARC Python wrapper
nordugrid-arc-java	ARC Java wrapper
nordugrid-arc-cache-service	ARC cache service
nordugrid-arc-janitor*	ARC dynamic installation of runtime environments
nordugrid-arc-aris*+	ARC LDAP information service
nordugrid-arc-isis	ARC isis service
ARC SE	
nordugrid-arc-chelonia	ARC chelonia service
nordugrid-arc-hopi	ARC hopi service
ARC IS	
nordugrid-arc-egiis+!	ARC EGIIS service
Security	
nordugrid-arc-gridmap-utils*!	NorduGrid authorization tools
nordugrid-arc-ca-utils*!	NorduGrid authentication tools
Monitoring	
nordugrid-arc-ldap-monitor	ARC LDAP monitor service
nordugrid-arc-ws-monitor	ARC WS monitor service
Documentation	
nordugrid-arc-doc	ARC documentation

Figure 3.1: **ARC packages**: the table shows a brief description of each package and the components they belong to. Packages marked with "!" are mandatory to have a working functionality. Packages marked with "*" are automatically installed by ARC-CE nordugrid-arc-compute-element metapackage, packages marked with "+" are automatically installed by ARC Infosys nordugrid-arc-information-index metapackage

and a LDAP certificate request with

```
grid-cert-request -service ldap -host <my.host.fqdn>
```

and send the request(s) to the NorduGrid CA for signing.

3.3.1 Installing host certificates

Once an host certificate is obtained from a CA, it has to be installed for the CE to use it.

When generating a certificate, two files will be created: a certificate file (public), tipically `hostcert.pem`; and a key file (private), tipically `hostkey.pem`.

Installation is as follows:

1. Copy the two files `hostcert.pem` and `hostkey.pem` into the standard ARC location: `/etc/grid-security`.
2. Both files must be owned by root.
3. The private key (`hostkey.pem`) must be readable only by root.
4. The two files MUST NOT have executable permissions.
5. The key file MUST NOT be password protected. This is especially important if a tool other than `grid-cert-request` was used.

If the ARC services will be run as a different user than root, then these files should be owned and accessible by this other user.

3.3.2 Installing custom CA certificates

If you're planning to install custom certificates such as the one provided by InstantCA (See **2.3**, *Certificates*) then the files must usually be copied into the `/etc/grid-security/certificates/` directory.

3.3.3 Authentication Policy

The credential-level authentication policy is just a decision on which certificates the computing element will accept. Only those users will be able to connect to the CE whose CAs are installed. (This does not mean they will be authorized to submit jobs, but at least they can establish the connection.) It is strongly advised to obtain a certificate from each CA by contacting it. To simplify this task, the NorduGrid Downloads area has a non-authoritative collection of CA credentials approved by EUGridPMA. As soon as you decide on the list of trusted certificate authorities, you simply download and install the packages containing their public keys and certificates. Before installing any CA package, you are advised to check the credibility of the CA and verify its policy!

> **Example** If your host certificate is issued by the NorduGrid CA, and your user has a certificate issued by the Estonian CA, and she is going to transfer files between your site and Slovakia, you need the NorduGrid, Estonian and Slovak CA credentials.

3.3.4 Revocation lists

The Certificate Authorities are responsible for maintaining lists of revoked personal and service certificates, known as CRL (Certificate Revocation List). It is the CE administrator responsibility to check the CRLs regularly and deny access to Grid users presenting a revoked certificate. Outdated CRL will render your site unuseable. A tool called `fetch-crl` exists to get the latest CRLs, which can be installed from the `fetch-crl` package (this package is not provided by NorduGrid). The tool is intended to run as a cron job. More information can be found here: `http://vdt.cs.wisc.edu/components/fetch-crl.html`

3.3.5 Authorization policy

The authorization policy is a decision on which grid users or groups of grid users (Virtual Organizations) are allowed to use your resource. Configuration of this will be discussed in the following sections: Section **4.4.1**, *Access control: users, groups, VOs* and Section **6.13**, *Structure of the grid-mapfile*.

Chapter 4

Configuration

This section leads through the following steps:

1. Prepare the system to run ARC services (Section **4.1**, *Preparing the system*)
2. Configure a basic CE (Section **4.2**, *Configuration file formats* and Section **4.3**, *Setting up a basic CE*)
3. Make it production-ready (Section **4.4**, *Production CE setup*)
4. Add optional features (Section **4.5**, *Enhancing CE capabilities*)

4.1 Preparing the system

4.1.1 Users and groups

ARC services are run by the `root` user by default, and this is the most convenient way for normal operation. But it is also possible to run it as a non-privileged user (see Section **4.1.3**, *Permissions*).

Users accessing the grid have a *grid identity* (see Section **1.5**, *Security on the Grid*) and will submit and run jobs on different physical machines. In ARC, each grid identity is mapped to a local UNIX user on the front-end machine (the one that runs A-REX) and eventually on the machine actually performing the job (worker nodes, managed by the LRMS). Hence, one or more local UNIX users need to be created in the system, to run the jobs submitted by grid clients.

It is possible to map all grid users to the same local user. For a basic CE setup, this will be sufficient. Later however for security reasons it is better to have a pool of local users to choose from, or the have actual local users for each grid user. To anticipate more users in the future, it is a good practice to create a dedicated local group for these mapped users, so that is possible to use local UNIX authorization methods to restrict the grid accounts.

For the basic CE setup, let's create a new group called **grid** and new user called **griduser1** that belongs to this group. Later more users can be created.

More advanced user configuration setups are discussed in Section **4.4.1**, *Access control: users, groups, VOs.*

4.1.2 Disk, partitioning, directories

The ARC CE uses separate directories to store the data files of the jobs, the metadata about the jobs, and the cached input files. It also requires a directory with the installed CA certificates and optionally can use a directory of runtime environments.

Figure 4.1 shows these directories, Table 4.1 summarizes how these directories should be configured.

Some of these directories are suggested to be local to the front-end, other can be on shared or networked filesystems on external storage. The following is a description of the important directories for ARC CE. Note: some of them are **Required** for the ARC CE to work.

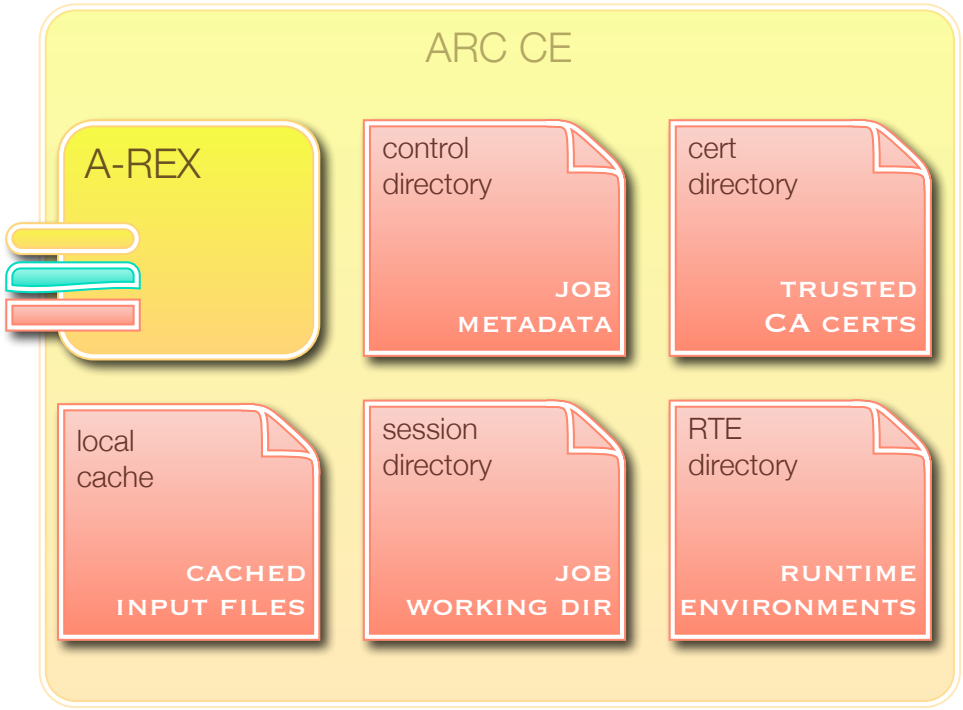


Figure 4.1: The directories on an ARC CE

Control Directory (CD) [Required] contains all the informations about jobs handled by the A-REX, such as job specification files and LRMS submission scripts. The information provider scripts also uses this directory to get information about jobs. This directory is heavily accessed by the A-REX, hence it should not be on a slow remote storage.

Session Directory (SD) [Required] contains the executable and data files of the jobs. This is where the jobs run, and this is the only area where they can produce results. Each job is assigned a unique directory within the session directory. This is usually shared among the worker nodes and the frontend, and can be remote for the frontend also. (See also Section **6.16**, *Using a scratch area*.)

Grid Certificates Directory [Required] contains the certificates of and other information about the trusted CAs. It is usually located at `/etc/grid-security/certificates`. (For setup instructions, see Section **3.3**, *Installation of certificates*.)

Cache Directory [Optional] can be used to cache downloaded input files, so if a new job requires the same file, it doesn't have to be downloaded again. Can reside on a shared filesystem. Caching is discussed in sections Section **4.4.3**, *Enabling the cache* and Section **6.5**, *Cache*.

Runtime Environments Scripts directory [Optional] contains special scripts that setup a particular runtime enviroment for a job to access. These include environment variables and software selections. Can reside on a shared filesystem. Runtime Environments are explained in Section **4.5.2**, *Runtime Environments*.

When partitioning disks and connecting shared storage, keep in mind the following things:

- The control directory (CD) is frequently accessed by the CE, so it is strongly advised to have it on a local hard disk. It can, however, grow pretty much with the number of jobs, so it is better to allocate a separate partition for it. The amount of data per job is around 50-100kb.
- The session directory (SD) stores all the executables, input and output files, and intermediate results of the jobs. It should be on a separate partation or even on a remote storage.

For more details please refer to sections Section **6.16**, *Using a scratch area*, Section **4.4.3**, *Enabling the cache*.

The ARC suggested setup for these directories is summarized in table 4.1.

Directory	Suggested Location	Example	Required?
sessions directory	NFS or shared FS, can be also on a separate disk partition	`/var/spool/arc/session`	Required
control directory	local to the front-end, also in a separate disk partition	`/var/spool/arc/control`	Required
CA certificates	local to the front-end	`/etc/grid-security/certificates`	Required
RTE scripts	NFS or shared FS	`/SOFTWARE/runtime`	Optional
cache directory	local, NFS, local and published via NFS	`/var/spool/arc/cache`	Optional

Table 4.1: Summary of ARC CE directories setup

4.1.3 Permissions

By default, the ARC services are run by root. In this case the control directory (CD) and the session directory (SD) should be writable, readable and executable by the root user, and then the A-REX will set all the other permissions as needed.

In case the ARC services should be run as a non-privileged (non-root) user, they cannot modify permissions of directories as easily. After the grid users are mapped to local users, they have to be able to access the job's session directory, hence the suggested setup is:

- put all the local users into the same group (e.g. `grid`)
- to set group ownership of the SD to this group
- the SD has to be writable, readable and executable by members of this group
- the SD and the CD has to be writable, readable and executable by the user running the ARC services

The host credentials need to have special permissions (see Section **3.3**, *Installation of certificates*).

4.1.4 Networking

DNS Requirements For the ARC middleware, the frontend has to have a public IP and a Fully Qualified Domain Name (FQDN) in order to join an indexing service and thus the grid (more on this on chapter Section **4.4.4**, *How to join the grid: registering to an index service*). This means that a reverse DNS lookup for the frontend's IP has to return the FQDN.

Basic networking recommendations are the following:

- Make sure your frontend has a FQDN. Issuing `hostname -f` should print it.
- In the `/etc/hosts` file, make sure that the FQDN of your machine comes **first**, before other network names. Example: if `130.235.185.195` is the IP address and `gridtest.hep.lu.se` is the FQDN assigned to it, `/etc/hosts` should look like:

```
130.235.185.195 gridtest.hep.lu.se gridtest
```

  while the following could lead to problems:

```
# wrong!
130.235.185.195 gridtest gridtest.hep.lu.se
```

Firewalls ARC-CE needs the following incoming and outgoing ports to be opened:

- For the web service interface: HTTP(s), default 80 and 443
- For MDS-LDAP infopublishing, default 2135 (see also Section **4.3.5**, *The [infosys] section: the local information system*)
- For the pre-web service interface: GridFTP,
  - default 2811
  - a range of ports for GridFTP data channels, typically 9000-9300
- For HTTPg, default 8443 (outgoing only)
- For SMTP, default 25 (outgoing only)
- For NTP, default 123 (outgoing only, in case NTP is used for time synchronisation, see **2**, *Requirements*)

Most ports, including 2135 and 2811, are registered with IANA and should normally not be changed. The ports for GridFTP data channels can be chosen arbitrary, based on following considerations: gridftpd by default handles 100 connections simultaneously; each connection should not use more than 1 additional TCP port. Taking into account that Linux tends to keep ports allocated even after the handle is closed for some time, it is a good idea to triple that amount. Hence about 300 data transfer ports should be enough for the default configuration. Typically, the range of ports from 9000 to 9300 is being opened. Remember to specify this range in the ARC configuration file (see Section **4.2**, *Configuration file formats*, `globus_tcp_port_range` attribute) later on.

For using legacy Globus components it is also worth to read information at this URL: `http://dev.globus.org/wiki/FirewallHowTo`

Other network related Internal cluster nodes (i.e. LRMS nodes) are NOT required to be fully available on the public internet (however, user applications may require it). For information about publishing nodes network connectivity please refer to Section **4.3.5.1**, *The [cluster] section: information about the host machine.*

4.1.5 Security considerations

SELinux If the system uses SELinux, the startup script should be usually able to create profiles for the services. If any problem in connectiong or starting up services arises, please set SELinux in permissive mode.

AppArmor On Ubuntu and Debian machines AppArmor profiles have been reported to prevent infosystem to start. AppArmor profiles are currently not shipped for ARC components. Therefore for the moment being:

- Remove `/etc/apparmor.d/usr.sbin.slapd` and restart AppArmor.
- If the above doesn't exist or doesn't help, disable AppArmor completely or put all the profiles in complain mode*.

4.2 Configuration file formats

Configuration of ARC can be done with a single configuration file usually located at `/etc/arc.conf`.

This configuration file format is fully compatible with the one for ARC middleware version 0.8.x.

> ⋆ If you have a legacy file from an ARC 0.8.x version,
> you can directly use that file for the new A-REX-based ARC CE.

*`https://help.ubuntu.com/community/AppArmor`

Using the the `arc.conf` is sufficient for the majority of use cases, however there is a possibility to use a lower-level XML-based configuration format (and a corresponding higher-level INI format) in special cases. For more details, see Section **6.10**, *The XML and the INI configuration formats.*

4.2.1 Structure of the `arc.conf` configuration file

An ARC legacy configuration file is a text file containing **sections** and related **commands**.

Each **section** identifies one or more components/features of ARC, and **commands** are used to modify the behaviour of these component/features.

A **section name** is sourrounded by square brackets and can contain slashes. Names after the slashes identify subsections. Examples:

```
[cluster]
[infosys]
[infosys/glue12]
[queue/fork]
[infosys/cluster/registration/toPGS1]
```

As a general rule, a section name containing a subsection has to appear **after** its section. Examples in Figure 4.2.

```
...
[infosys]
...
[infosys/glue12]
...
[queue/fork]
...
[infosys/cluster/registration/toPGS1]
...
```

Correct

```
...
[infosys/cluster/registration/toPGS1]
...
[infosys/glue12]
...
[infosys]
...
[queue/fork]
...
```

Wrong

Figure 4.2: Ordering of section names.

A **configuration command** is a one-line `command="value"` expression. Examples:

```
hostname="gridtest.hep.lu.se"
nodecpu="2"
resource_location="Lund, Sweden"
mail="gridmaster@hep.lu.se"
```

Comments can be added one per line by putting a `#` at the beginning of the line.

A section starts with a section name and ends at another section name or if the end of the configuration file is reached. Configuration commands always belong to one section.

Here is an overall example:

```
# this is a comment, at the beginning of the [common] section
[common]
hostname="piff.hep.lu.se"
x509_user_key="/etc/grid-security/hostkey.pem"
x509_user_cert="/etc/grid-security/hostcert.pem"
x509_cert_dir="/etc/grid-security/certificates"
gridmap="/etc/grid-security/grid-mapfile"
lrms="fork"

# since there is a new section name below, the [common] section ends
# and the grid-manager section starts
```

```
[grid-manager]
user="root"
controldir="/tmp/control"
sessiondir="/tmp/session"
# cachedir="/tmp/cache"
debug="3"

# other commands...

[queue/fork]

# other commands till the end of file.
# This ends the [queue/fork] section.
```

4.2.2 Description of configuration items

In the descriptions of commands, the following notation will be used:

command=*value [value]* – where the values in square brackets *[...]* are *optional*. They should be inserted without the square brackets!

A pipe "|" indicates an exclusive option. Example:

securetransfer=*yes|no* – means that the value is either yes or no.

For a complete list and description of each configuration item, please refer to Section **6.1**, *Reference of the arc.conf configuration commands*.

The configuartion commands are organized in sections. The following is a description of the main **mandatory** sections and of the components and functionalities they apply to, in the order they should appear in the configuration file. These are needed for a minimal and basic functionalities (see Section **4.3**, *Setting up a basic CE*).

[common] Common configuration affecting networking, security, LRMS. These commands define defaults for all the ARC components (A-REX, GridFTPd, ARIS), which can be overridden by the specific sections of the components later. Always appears at the beginning of the config file.

Discussed in Section **4.3.2**, *The [common] section*.

[group] This section and its subsections define access control mappings between grid users and local users. Applies to all ARC components. Usually follows the [common] section. If there are [vo] sections, they should come before the [group] section.

Discussed in Section **4.4.1**, *Access control: users, groups, VOs*.

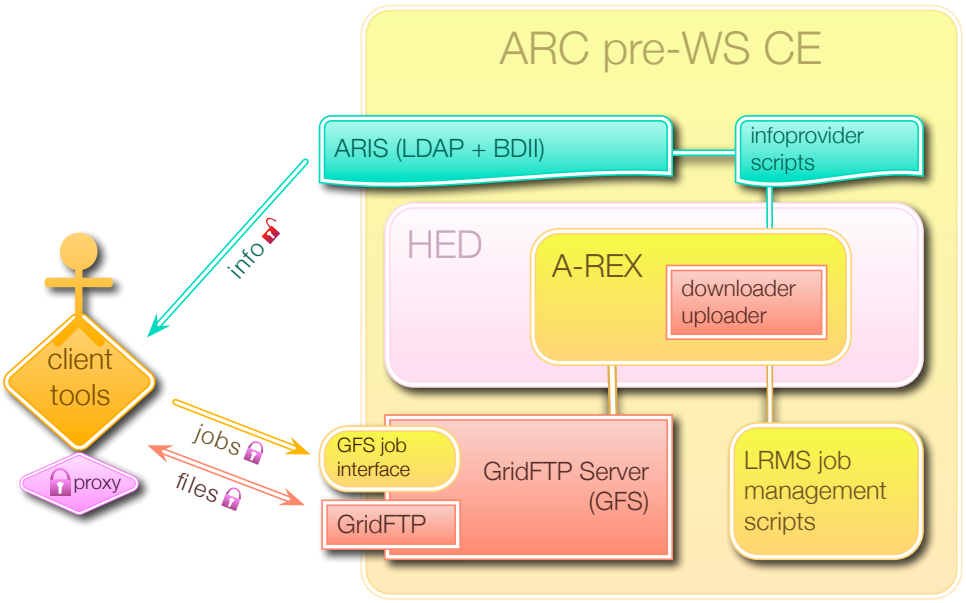
If no access control is planned (for example for tests) this section can be omitted but the administrator must manually edit the grid-mapfile (see Section **6.13**, *Structure of the grid-mapfile*)

[grid-manager] This section configures the A-REX, including job management behavior, directories, file staging and logs.

Discussed in Section **4.3.3**, *The [grid-manager] section: setting up the A-REX and the arched*.

[gridftpd] This section configures the GridFTPd, which is the server process running the GridFTP protocol. Its subsections configure the different plugins of the GridFTPd, in particular the job submission interface: **[gridftpd/jobs]**.

Discussed in Section **4.3.4**, *The [gridftpd] section: the job submission interface*.



[infosys] This section configures the local information system (ARIS) and the information provider scripts. (This section also can be used to configure an information index server, see [**?**].) The commands affects the data published by the information system, the behaviour of the publishing server and its networking options. The subsections configure registration to information index servers, and extra informations for different information schemas.

Discussed in Section **4.3.5**, *The [infosys] section: the local information system.*

[cluster] Configures the A-REX information provider scripts. The commands here affects the data published by the local information system, mostly regarding the front-end machine. Must appear after the `[infosys]` section.

Discussed in Section **4.3.5.1**, *The [cluster] section: information about the host machine*

[queue/queuename] Configures the A-REX information provider scripts. At least [queue/...] section must exist. The commands here affects the data published by the information system, mostly regarding the LRMS queues A-REX is serving. Must appear after the `[infosys]` section.

Discussed in Section **4.3.5.2**, *The [queue/fork] section: configuring the fork queue.*

Generic commands These commands specify common defaults in the `[common]` section, and also can be used to set different values per component in the following sections: `[grid-manager]`, `[gridftpd]` and its subsections and `[infosys]`.

logfile=*path* – where the logs will be written.

pidfile=*path* – where the PID of the process will be written.

debug=*number* – specifies the level of logging from 5 (DEBUG) to 0 (FATAL).

4.3 Setting up a basic CE

A basic CE is the starting point of every ARC setup. A basic CE is a stand-alone machine ready to accept job submission. A basic CE will not be connected to an information index, so clients will have to explicitly specify its job submission interface URL to connect to. This chapter will show a basic configuration of the main sections seen in chapter Section **4.2.2**, *Description of configuration items.*

Please make sure all the steps in chapter Section **4.1**, *Preparing the system* are done before proceeding.

The basic CE will have **fork** as an LRMS, which will allow the machine to process jobs in the environment provided by the operating system of the front-end machien. Connection to real LRMSes are discussed in Section **4.4.2**, *Connecting to the LRMS*.

4.3.1 Creating the arc.conf file

ARC will by default search for its configuration file in the following location:

```
/etc/arc.conf
```

The minimal configuration file described in the following is usually installed here:

```
/usr/share/arc/example/arc_computing_element.conf
```

or it can be downloaded from the ARC Configuration Examples web page[†].

Copy this file into `/etc` with the name `arc.conf`, then modify its contents following the instructions below.

4.3.2 The [common] section

The `[common]` section maintains informations that will be used by any subsystem of the CE. It has to appear as the first item in the configuration file.

A minimal configuration for this section is shown here:

```
[common]
x509_user_key="/etc/grid-security/hostkey.pem"
x509_user_cert="/etc/grid-security/hostcert.pem"
x509_cert_dir="/etc/grid-security/certificates"
gridmap="/etc/grid-security/grid-mapfile"
lrms="fork"
```

Here we specify the path of the host's private key and certificate, the directory where the certificates of the trusted Certificate Authorities (CAs) are located, the path of the grid map file, which defines mapping of grid users to local users, and the name of the default lrms, which is "fork" in the basic case, when we only want to use the frontend as a worker node, not a real cluster.

For details about these configuration commands, please see Section **6.1.1**, *Generic commands in the [common] section*

For the basic CE, let's create a "grid map file" which looks like this:

```
"/DC=eu/DC=KnowARC/O=Lund University/CN=demo1" griduser1
"/DC=eu/DC=KnowARC/O=Lund University/CN=demo2" griduser1
"/DC=eu/DC=KnowARC/O=Lund University/CN=demo3" griduser1
```

4.3.3 The [grid-manager] section: setting up the A-REX and the arched

The `[grid-manager]` section configures `A-REX` and `arched`. Its commands will affect the behaviour of the startup scripts and the `A-REX` and `arched` processes.

A sample section would look like this:

```
[grid-manager]
user="root"
controldir="/tmp/jobstatus"
sessiondir="/tmp/grid"
```

[†] `http://www.nordugrid.org/arc/configuration-examples.html`

```
debug="3"
logfile="/tmp/grid-manager.log"
pidfile="/tmp/grid-manager.pid"
mail="grid.support@somewhere.org"
joblog="/tmp/gm-jobs.log"
```

Here we specify which user the A-REX should be run as, where should be the directory for the job's metadata (the control dir) and data (the session dir), what level of log message we want, where should be the log file and where should the process ID of the arched daemon be written. We also specify an e-mail contact address and the path of the "joblog" file, which will contain informations about the job's lifecycle.

For details about these configuration commands, please see Section **6.1.11**, *Commands in the [grid-manager] section*

4.3.4 The [gridftpd] section: the job submission interface

Currently, the production level job submission interface uses the **gridftp** protocol which is served by the GridFTP Server (GFS) running on the frontend.

The `[gridftpd]` section configures the behaviour of the gridftpd daemon and its startup scripts.

A sample section for a basic CE is the following:

```
[gridftpd]
user="root"
debug="3"
logfile="/tmp/gridftpd.log"
pidfile="/tmp/gridftpd.pid"
port="2811"
allowunknown="no"
```

Here we specify which user the GridFTP server should run as, the verbosity of the log messages, the path of the logfile and the pidfile, the port of the GridFTP server, and that only "known" users (specified in the grid map file) should be allowed to connect.

For a minimal ARC CE to work, we need the configure the job interface with setting up the "job plugin" of the GridFTP server in a configuration subsection:

`[gridftpd/jobs]` controls how the virtual path /jobs for job submission will behave. These paths can be thought as those of a UNIX mount command. The name *jobs* itself is not relevant, but the contents of the section and especially the `plugin` command determine the path behaviour.

For a minimal CE to work, it is sufficient to configure the following:

```
[gridftpd/jobs]
path="/jobs"
plugin="jobplugin.so"
allownew="yes"
```

Here we specify the virtual path where the job plugin will sit, the name of the library of the plugin, and that new jobs can be submitted (turning `allownew` to "no" would stop accepting new jobs, but the existing jobs would still run.)

For details about these configuration commands, please see Section **6.1.4**, *Commands in the [gridftpd] section*

As GridFTPd interface is planned to be phased out and replaced by the web service interface, no big changes will be done in the future. For a detailed description of GridFTPd configuration, please refer to the official nordugrid manual [**?**].

4.3.5 The [infosys] section: the local information system

The `[infosys]` section and its subsections control the behaviour of the information system. This includes:

- configuration of ARIS and its infoproviders
- customization of the published information
- configuration of the `slapd` server to publish information via LDAP
- configuration of BDII to generate ldif trees for LDAP
- selection of the LDAP schema(s) to publish
- registration to an EGIIS index service (see Section **4.4.4**, *How to join the grid: registering to an index service*)
- running a EGIIS IS (not covered in this manual, please refer to [])

After this section, several subsections will appear as well as some other sectons which are related to the information system, such as `[cluster]` and `[queue/...]` sections. More on these will be explained later.

A sample configuration for a basic CE would be the following:

```
[infosys]
user="root"
overwrite_config="yes"
port="2135"
debug="1"
slapd_loglevel="0"
registrationlog="/tmp/inforegistration.log"
providerlog="/tmp/infoprovider.log"
provider_loglevel="2"
```

Here we specify which user the slapd server, the infoproviders, the BDII and the registration scripts should run, then we specify that we want the low-level slapd configs to be regenerated each time, then the port number, the debug verbosity of the startup script, the slapd server and the infoproviders, and the logfiles for the registration messages and the infoprovider messages.

For details about these configuration commands, please see Section **6.1.5**, *Commands in the [infosys] section.*

4.3.5.1 The [cluster] section: information about the host machine

This section has to follow the `[infosys]` section and it is used to configure the information published about the host machine running ARC CE.

A sample configuration can be seen below:

```
[cluster]
cluster_alias="MINIMAL Computing Element"
comment="This is a minimal out-of-box CE setup"
homogeneity="True"
architecture="adotf"
nodeaccess="inbound"
nodeaccess="outbound"
```

Here we specify the alias of the cluster, a comment about it, that the worker nodes are homogenious, that we want infoprovider scripts to determine the architecture automatically on the frontend ("adotf"), and that the worker nodes has inbound and outbound network connectivity.

For details about these configuration commands, please see Section **6.1.8**, *Commands in the [cluster] section.*

4.3.5.2 The [queue/fork] section: configuring the fork queue

Each `[queue/queuename]` section configures the information published about computing queues. At least one queue must be specified for a CE to work. In this chapter a configuration for the **fork** backend will be shown.

The fork backend is just a simple execution environment provided by the means of the underlying operating system, that is,usually a shell with the standard linux environment variables provided to the mapped UNIX user.

A special section name [queue/fork] is used to configure such information, some of its commands can be used for any queue section, some are specific for the fork queue. More about this will be explained in Section **4.4.2**, *Connecting to the LRMS*.

A minimal CE configuration for this section would look like this:

```
[queue/fork]
name="fork"
fork_job_limit="cpunumber"
homogeneity="True"
scheduling_policy="FIFO"
comment="This queue is nothing more than a fork host"
nodecpu="adotf"
architecture="adotf"
```

Here we specify that this is a "fork" queue, that the number of allowed concurent jobs should equal the number of CPUs, that the queue is homogenious, the scheduling policy, an informative comment, and that the type of the cpu and the architecture should be determined automatically on the frontend. The only fork-specific command is the `fork_job_limit` command, the others are for the other backends also. See sections Section **4.4.2**, *Connecting to the LRMS* and Section **6.1.9**, *Commands in the [queue] subsections*.

4.3.6 A basic CE is configured. What's next?

A basic CE is now set. To test its functionality, it must be started first. Please refer to Section **5.1.2**, *Starting the CE* to start the CE. If none of the startup scripts give any error, the testing can be started. Please follow the testing suggestions in Section **5.2**, *Testing a configuration*.

If everything works as expected, the next step is the turn the basic CE into a production level CE: connecting it to the LRMS, turning on input file caching, and register it to an information index service. Please follow the instructions in Section **4.4**, *Production CE setup*.

For some additional (optional) features, please proceed to Section **4.5**, *Enhancing CE capabilities*.

4.4 Production CE setup

Once a basic CE is in place and its basic functionalities have been tested, these things are usually needed to make it production-ready:

Configure access control to streamline the maintenance of the authentication and authorization of users, VOs and authorization groups should be defined and the `nordugridmap` tool should be utilized to generate the grid map file automatically. See Section **4.4.1**, *Access control: users, groups, VOs*.

Connect to the LRMS to be able to use the underlying batch system, ARC support several famous clustering and load balancing systems such as Torque/PBS, Sun Grid Engine, LSF, and others. See Section **4.4.2**, *Connecting to the LRMS*.

Enabling the cache to keep a copy of the downloaded input files in case the next job needs the same, which greatly decreases wait time for jobs to start. See Section **4.4.3**, *Enabling the cache*

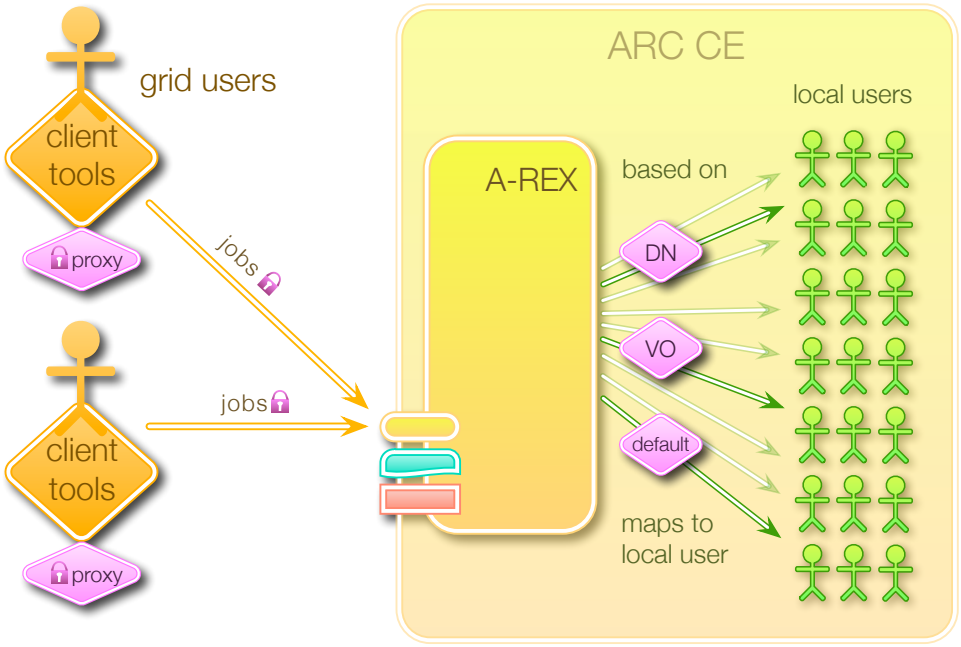


Figure 4.3: The A-REX maps the grid users to local users based on information about their identity and Virtual Organization membership. It's also possible to do default mapping.

Register to an index service NorduGrid provides an index service that will publish the CE to all the grid clients that have access to the NorduGrid network. In this way the CE will be part of the GRID. See Section **4.4.4**, *How to join the grid: registering to an index service*.

Accounting the A-REX is capable to send usage records to the SGAS accounting service. See Section **??**, **??**.

Monitoring Nagios plugins exist for monitoring the ARC Computing Element. See Section **4.4.7**, *Monitoring the ARC CE: Nagios probes*.

4.4.1 Access control: users, groups, VOs

The grid mappings between grid users and local unix accounts are listed in the so-called `grid map file`, usually located in the directory `/etc/grid-security/`. While this text file can be edited by hand this is not advisible in production environments. To ease the security administrator job, NorduGrid provides a collection of scripts and cron jobs that automatically keeps the local grid map files synchronized to a central user database. If the CE has to join NorduGrid, it is suggested to install the `nordugrid-arc-gridmap-utils` package from the NorduGrid Downloads area or EMI repository, see Chapter **3**, *Installation* for details. If you installed the package, then you should edit the `[groups]` and `[vo]` sections in the configuration file and optionally the location of the file representing local list of mappings (can have any name, but usually called `/etc/grid-security/local-grid-mapfile`). For the description of the grid map file, please refer to Section **6.13**, *Structure of the grid-mapfile*.

The two sections `[group]` and `[vo]` configure automatic mapping of GRID identities to local UNIX users and basic access control policies:

`[vo]` defines Virtual Organizations (VOs). A VO is a simple way of mapping sets of users belonging to different (real) organizations and, for example, willing to use the same set of software. A common use of this section is to include users published by VOMS servers [**?**]. `[vo]` sections can be referred by `[group]` sections. If this happens, it is important that the `[vo]` definition appears **before** the `[group]` section that refers to it.

`[group]` defines *authorization rules* to access the CE for users or set of users defined by [vo] sections.

The configuration presented here is sufficient for a simple production setup where the identities are known or are already contained in a file or a collection of files, eventually located and updated remotely.

4.4.1.1 [vo] configuration commands

The following is a sample [vo] section for a minimal CE:

```
[vo]
id="vo_1"
vo="TestVO"
source="file:///etc/grid-security/local-grid-mapfile"
mapped_unixid="griduser1"
require_issuerdn="no"
```

We define a VO here with the name of `TestVO` and the id of `vo_1`, the list of members comes from a URL (which here points to a local file, see example below), and all members of this VO will be mapped to the local user `griduser1`.

Here's an example of the file with the list of members:

```
"/DC=eu/DC=KnowARC/O=Lund University/CN=demo1"
"/DC=eu/DC=KnowARC/O=Lund University/CN=demo2"
"/DC=eu/DC=KnowARC/O=Lund University/CN=demo3"
"/DC=eu/DC=KnowARC/O=Lund University/CN=demo4"
"/DC=eu/DC=KnowARC/O=Lund University/CN=demo5"
```

For more configuration options, please see Section **6.1.2**, *Commands in the [vo] section.*

To generate the actual grid map file from these [vo] settings, we need the `nordugridmap` utility, described below.

4.4.1.2 Automatic update of the mappings

The package `nordugrid-arc-gridmap-utils` contains a script to automatically update user mappings (usually located in `/usr/sbin/nordugridmap`). It does that by fetching all the sources in the `source` commands and writing their contents adding the mapped user `mapped_unixid` in the grid-mapfile and each file specified by the `file` command. The script is executed from time to time as a cron job.

4.4.1.3 [group] configuration commands

`[group]` defines *authorizations* for users accessing the grid.

There can be more than one group in the configuration file, and there can be subsections identified by the group name such as `[group/users]`.

For a minimal CE with no authorization rules, it is sufficient to have something like the following, preceeded with the `[vo]` section previously defined in this chapter:

```
[group/users]
name="users"
vo="TestVO"
```

where the `name` could be omitted and then would be automatically taken from the subsection name.

For more about authorization, please read Section **6.1.3**, *Commands in the [group] section.*

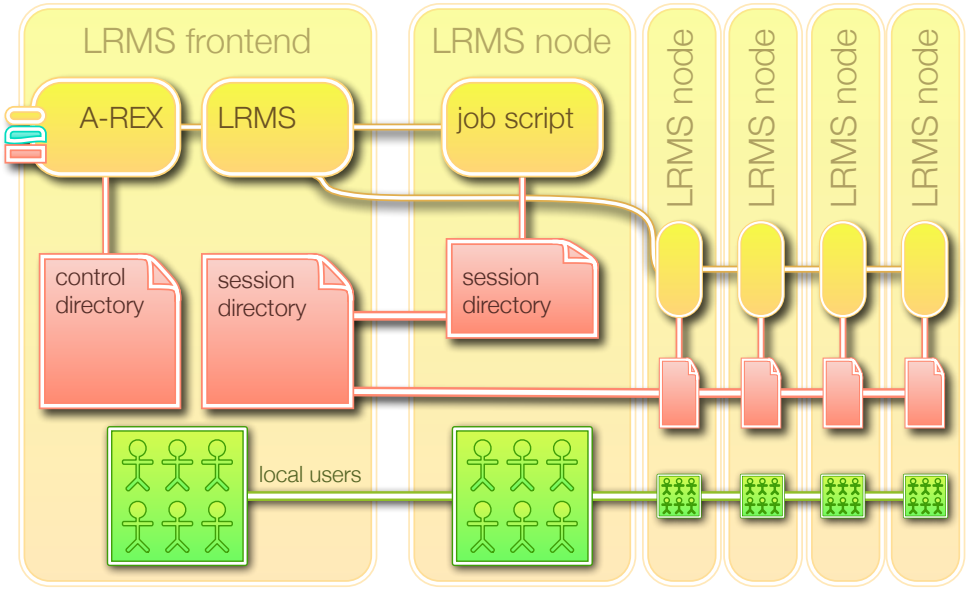


Figure 4.4: The LRMS frontend and the nodes sharing the session directory and the local users

4.4.2 Connecting to the LRMS

One of the features of Grid Middleware is that leverages the user from learning all the different commands needed to submit a job to a queue, thus still taking advantage of all the features load levelling and clustering system provide. A-REX supports several Local Resource Management Systems, with which it interacts by several backend scripts.

Connecting A-REX to one of these LRMS involves the following steps:

1. creation of shared directories between A-REX, the LRMS frontend and its working nodes. It might involve setup of shared filesystems such as NFS or similar.
2. configuration of the behaviour of a-rex with respect to the shared directories in the `[grid-manager]` section.
3. configuration of the following `arc.conf` sections: `[common]`, `[grid-manager]`, `[queue/*]`.

In the `[common]` section the name of the LRMS has to be specified:

`lrms`=*`default_lrms_name [default_queue_name]`* – specifies the name of the LRMS and optionally the queue.

The following `[grid-manager]` configuration commands affect how A-REX interacts with the LRMS:

`gnu_time`=*`path`* – path to *time* utility.

`tmpdir`=*`path`* – path to directory for temporary files. Default is `/tmp`.

`runtimedir`=*`path`* – path to directory which contains *runtimenvironment* scripts.

`shared_filesystem`=*`yes|no`* – if computing nodes have an access to session directory through a shared file system like NFS. if set to "*no*", this means that the computing node does not share a filesystem with the frontend. In this case the content of the SD is moved to a computing node using means provided by the LRMS. Results are moved back after the job's execution in a similar way. Sets the environment variable `RUNTIME\_NODE\_SEES\_FRONTEND`

`scratchdir`=*`path`* – path on computing node where to move session directory before execution. If defined should contain the path to the directory on computing node which can be used to store a job's files during execution. Sets the environment variable `RUNTIME\_LOCAL\_SCRATCH\_DIR` .

shared_scratch=*path* – path on frontend where ***scratchdir*** can be found. if defined should contain the path corresponding to that set in `scratchdir` as seen on the **frontend** machine. Sets the environment variable `RUNTIME\_FRONTEND\_SEES\_NODE` .

nodename=*command* – command to obtain hostname of computing node.

For additional details, see Section **6.1.11.9**, *Substitutions in the command arguments* and Section **6.16**, *Using a scratch area*.

Each LRMS has his own peculiar configuration options.

4.4.2.1 PBS

The Portable Batch System (PBS) is one of the most popular batch systems. PBS comes in many flavours such as OpenPBS (unsupported), Terascale Open-Source Resource and QUEue Manager (TORQUE) and PBSPro (currently owned by Altair Engineering). ARC supports all the flavours and versions of PBS.

Recommended batch system configuration PBS is a very powerful LRMS with dozens of configurable options. Server, queue and node attributes can be used to configure the cluster's behaviour. In order to correctly interface PBS to ARC (mainly the information provider scripts) there are a couple of configuration REQUIREMENTS asked to be implemented by the local system administrator:

1. The computing nodes MUST be declared as cluster nodes (job-exclusive), at the moment time-shared nodes are not supported by the ARC setup. If you intend to run more than one job on a single processor then you can use the virtual processor feature of PBS.
2. For each queue, you MUST set one of the max_user_run or max_running attributes and its value SHOULD BE IN AGREEMENT with the number of available resources (i.e. don't set the max_running = 10 if you have only six (virtual) processors in your system). If you set both max_running and max_user_run then obviously max_user_run has to be less or equal to max_running.
3. For the time being, do NOT set server limits like max_running, please use queue-based limits instead.
4. Avoid using the max_load and the ideal_load directives. The Node Manager (MOM) configuration file (`<PBS home on the node>/mom_priv/config`) should not contain any max_load or ideal_load directives. PBS closes down a node (no jobs are allocated to it) when the load on the node reaches the max_load value. The max_load value is meant for controlling time-shared nodes. In case of job-exclusive nodes there is no need for setting these directives, moreover incorrectly set values can close down your node.
5. Routing queues are now supported in a simple setup were a routing queue has a single queue behind it. This leverages MAUI work in most cases.
   Other setups (i.e. two or more execution queues behind a routing queue) cannot be used within ARC.

Additional useful configuration hints:

- If possible, please use queue-based attributes instead of server level ones (for the time being, do not use server level attributes at all).
- You may use the "acl_user_enable = True" with "acl_users = user1,user2" attribute to enable user access control for the queue.
- It is advisory to set the max_queuable attribute in order to avoid a painfully long dead queue.
- You can use node properties from the `<PBS home on the server>/server_priv/nodes` file together with the `resources_default.neednodes` to assign a queue to a certain type of node.

Checking your PBS configuration:

- The node definition can be checked by `<PBS installation path>/bin/pbsnodes -a`. All the nodes MUST have ntype=cluster.

- The required queue attributes can be checked as <PBS installation path>/bin/qstat -f -Q queuename. There MUST be a max_user_run or a max_running attribute listed with a REASONABLE value.

Configuration commands in `arc.conf` Below the PBS specific configuration variables are collected.

`lrms`=*"pbs"* – in the [common] section enables the PBS batch system back-end. No need to specify the flavour or the version number of the PBS, simply use the "pbs" keyword as LRMS configuration value.

For each grid-enabled (or grid visible) PBS queue a corresponding [queue/queuename] subsection must be defined. queuename should be the PBS queue name.

`pbs_bin_path`=*path* – in the [common] section should be set to the path to the qstat,pbsnodes,qmgr etc. PBS binaries.

`pbs_log_path`=*path* – in the [common] sections should be set to the path of the PBS server logfiles which are used by A-REX to determine whether a PBS job is completed. If not specified, A-REX will use the qstat command to find completed jobs.

For additional configuration commands, please see Section **6.1.13**, *PBS specific commands*.

Known limitations Some of the limitations are already mentioned under the PBS deployment requirements. No support for routing queues, difficulty of treating overlapping queues, the complexity of node string specifications for parallel jobs are the main shortcomings.

4.4.2.2 Condor

The Condor [**?**] system, developed at the University of Wisconsin-Madison, was initially used to harness free cpu cycles of workstations. Over time it has evolved into a complex system with many grid-oriented features. Condor is available on a large variety of platforms.

Recommended batch system configuration Install Condor on the Grid Manager (GM) node and configure it as a submit machine. Next, you must add the following to the node's Condor configuration (CONDOR_IDS can also be an environment variable):

```
MAIL = <ARC_install_prefix>/libexec/finish-condor-job
CONDOR_IDS = 0.0
```

The MAIL attribute will instruct Condor to run the specified program on job completion. The default on Condor is to run /bin/mail to notify the user, but in this case, it is A-REX that needs the notification. Therefore, /bin/mail is replaced with a program especially written for talking to A-REX.

CONDOR_IDS has to be 0.0, so that the above notification program can access the Grid job's session directories (needed to extract the job exit code from the Condor log).

Make sure that no normal users are allowed to submit Condor jobs from this node. For one thing, it would not work for the user, since Condor will try to notify A-REX instead of the job owner on job completion. If you don't allow normal user logins on the A-REX machine, then you don't have to do anything. If you for some reason want to allow users to log into the A-REX machine, simply don't allow them to execute the condor_submit program. This can be done by putting all local Unix users allocated to the Grid in a single group, e.g. 'griduser', and then setting the file ownership and permissions on condor_submit like this:

```
chgrp griduser $condor_location/bin/condor_submit
chmod 750 $condor_location/bin/condor_submit
```

Configuration commands in `arc.conf` The Condor-specific configuration commands:

`lrms`=*"condor"* – in the `[common]` section enables the Condor batch system back-end.

`condor_location`=*path* – in the `[common]` section should be set to the Condor install prefix (i.e., the directory containing Condor's bin, sbin, etc).

For additional configuration commands, please see Section **6.1.14**, *Condor specific commands.*

Known limitations Only Vanilla universe is supported. MPI universe (for multi-CPU jobs) is not supported. Neither is Java universe (for running Java executables). ARC can only send jobs to Linux machines in the Condor pool, therefore excluding other unixes and Windows destinations. The session directory must be on a network shared directory, visible from all worker nodes.

4.4.2.3 LoadLeveler

LoadLeveler(LL), or Tivoli Workload Scheduler LoadLeveler in full, is a parallel job scheduling system developed by IBM.

Recommended batch system configuration The back-end should work fine with a standard installation of LoadLeveler. For the back-end to report the correct memory usage and cputime spent, while running. LoadLeveler has to be set-up to show this data in the llq command. Normally this is turned off for performance reasons. It is up to the cluster administrator to decide whether or not to publish this information. The back-end will work whether or not this is turned on.

Configuration commands in `arc.conf` Only the two basic LRMS config options are relevant for LoadLeveler:

`lrms`=*"ll"* – in the `[common]` section enables the LoadLeveler batch system.

`ll_bin_path`=*path* – in the `[common]` section must be set to the path of the LoadLeveler binaries.

Known limitations There is at the moment no support for parallel jobs on the LoadLeveler back-end.

4.4.2.4 Fork

The Fork back-end is a simple back-end that interfaces to the local machine, i.e.: there is no batch system underneath. It simply forks the job, hence the name. The back-end then uses standard posix commands (e.g. ps or kill) to manage the job.

Recommended batch system configuration Since fork is a simple back-end and does not use any batch system, there is no specific configuration needed for the underlying system.

Configuration commands in `arc.conf` Only these commands are applied:

`lrms`=*"fork"* – in the `[common]` section enables the Fork back-end. The queue must be named `"fork"` in the `[queue/fork]` subsection.

`fork_job_limit`=*cpunumber* – sets the number of running grid jobs on the fork machine, allowing a multi-core machine to use some or all of its cores for Grid jobs. The default value is 1.

Known limitations Since Fork is not a batch system, many of the queue specific attributes or detailed job information is not available. The support for the "Fork batch system" was introduced so that quick deployments and testing of the middleware can be possible without dealing with deployment of a real batch system since fork is available on every UNIX box. The "Fork back-end" is not recommended to be used in production. The back-end by its nature, has lots of limitations, for example it does not support parallel jobs.

4.4.2.5 LSF

Load Sharing Facility (or simply LSF) is a commercial computer software job scheduler sold by Platform Computing. It can be used to execute batch jobs on networked Unix and Windows systems on many different architectures

Recommended batch system configuration Set up one or more LSF queues dedicated for access by grid users. All nodes in these queues should have a resource type which corresponds to the one of the the frontend and which is reported to the outside. The resource type needs to be set properly in the `lsb.queues` configuration file. Be aware that LSF distinguishes between 32 and 64 bit for Linux. For a homogeneous cluster, the `type==any` option is convenient alternative.

Example: In `lsb.queues` set one of the following:

```
RES_REQ = type==X86_64
RES_REQ = type==any
```

See the `-R` option of the `bsub` command man page for more explanation.

Configuration commands in `arc.conf` The LSF back-end requires that the following options are specified:

`lrms`=*"lsf"* – in the `[common]` section enables the LSF back-end

`lsf_bin_path`=*path* – in the `[common]` section must be set to the path of the LSF binaries

`lsf_profile_path`=*path* – must be set to the filename of the LSF profile that the back-end should use.

Furthermore it is very important to specify the correct architecture for a given queue in arc.conf. Because the architecture flag is rarely set in the xRSL file the LSF back-end will automatically set the architecture to match the chosen queue. LSF's standard behaviour is to assume the same architecture as the frontend. This will fail for instance if the frontend is a 32 bit machine and all the cluster resources are 64 bit. If this is not done the result will be jobs being rejected by LSF because LSF believes there are no useful resources available.

Known limitations Parallel jobs have not been tested on the LSF back-end.

The back-end does not at present support reporting different number of free CPUs per user.

4.4.2.6 SGE

Sun Grid Engine (SGE, Oracle Grid Engine, Codine) is an open source batch system maintained by Sun (Oracle). It is supported on Linux, and Solaris in addition to a numerous other systems.

Recommended batch system configuration Set up one or more SGE queues for access by grid users. Queues can be shared by normal and grid users. In case you want to set up more than one ARC queue, make sure that the corresponding SGE queues have no shared nodes among them. Otherwise the counts of free and occupied CPUs might be wrong. Only SGE versions 6 and above are supported.

Configuration commands in `arc.conf` The SGE back-end requires that the following options are specified:

`lrms`=*"sge"* – in the `[common]` section enables the SGE batch system back-end.

`sge_root`=*path* – in the `[common]` section must be set to SGE's install root.

`sge_bin_path`=*path* – in the `[common]` section must be set to the path of the SGE binaries.

`sge_jobopts`=*options* – in the `[queue/queuename]` section can be used to add custom SGE options to job scripts submitted to SGE. Consult SGE documentation for possible options. Example:

```
lrms="sge"
sge_root="/opt/n1ge6"
sge_bin_path="/opt/n1ge6/bin/lx24-x86"

...

[queue/long]
sge_jobopts="-P atlas -r yes"
```

For additional configuration commands, please see Section **6.1.18**, *SGE specific commands.*

Known limitations Multi-CPU support is not well tested. All users are shown with the same quotas in the information system, even if they are mapped to different local users. The requirement that one ARC queue maps to one SGE queue is too restrictive, as the SGE's notion of a queue differs widely from ARC's definition. The flexibility available in SGE for defining policies is difficult to accurately translate into NorduGrid's information schema. The closest equivalent of nordugrid-queue-maxqueuable is a per-cluster limit in SGE, and the value of nordugrid-queue-localqueued is not well defined if pending jobs can have multiple destination queues.

4.4.2.7 SLURM

SLURM is an open-source (GPL) resource manager designed for Linux clusters of all sizes. It is designed to operate in a heterogeneous cluster with up to 65,536 nodes. SLURM is actively being developed, distributed and supported by Lawrence Livermore National Laboratory, Hewlett-Packard, Bull, Cluster Resources and SiCortex

Recommended batch system configuration The backend should work with a normal installation using only SLURM or SLURM+moab/maui. Do not keep nodes with different amount of memory in the same queue.

Configuration commands in `arc.conf` The SLURM back-end requires that the following options are specified:

`lrms`=*"SLURM"* – in the `[common]` section enables the SLURM batch system back-end.

`slurm_bin_path`=*path* – in the `[common]` section must be set to the path of the SLURM binaries.

Known limitations If you have nodes with different amount of memory in the same queue, this will lead to misscalculations. If SLURM is stopped, jobs on resource will get canceled, not stalled. The SLURM backend is only tested with SLURM 1.3, it should however work with 1.2 as well.

4.4.3 Enabling the cache

The A-REX can cache input files, so that subsequent jobs requiring the same file don't have to wait for downloading it again: the cached file will be symlinked (or copied) into the session directory of the job (but only after the permissions of this user and the modification date of the file are checked).

Enabling caching is as simple as providing a directory path with the `cachedir` configuration command in the `[grid-manager]` section and turning on the cache cleaning mechanism with the `cachesize` command:

```
cachedir=path
cachesize=high_mark [low_mark]
```

Here `path` points to a directory which will be used by the A-REX to store the cached files. A-REX will create this directory when the first job is submitted, it should be owned by the same user as which the A-REX is running. The size of the cache directory is maintained by removing the least recently accessed

files. If the cache size exceeds a given percentage ("high mark") of the available space, the oldest files will be removed until the size goes below another given percentage ("low mark").

A sample section is shown here:

```
[grid-manager]
user="root"
controldir="/tmp/control"
sessiondir="/tmp/session"
mail="grid.support@somewhere.org"
joblog="/tmp/gm-jobs.log"
securetransfer="no"
cachedir="/tmp/cache"
cachesize="80 70"
```

It is possible to use more than one cache directory by simply specifing more than one `cachedir` command in the configuration file. When multiple caches are used, a new cache file will go to a randomly selected cache where each cache is weighted according to the size of the file system on which it is located (e.g. if there are two caches of 1TB and 9TB then on average 10% of input files will go to the first cache and 90% will go to the second cache).

By default the files will be linked into the session directory of the job. If we prefer to copy them (because e.g. the cache directory is not accessible from the worker nodes), we should add a dot (.) after the path:

```
cachedir=path "."
```

If the cache directory is accessible from the worker nodes but on a different path, then we can specify this path also:

```
cachedir=path link_path
```

With large caches mounted over NFS and an A-REX heavily loaded with data transfer processes, cache cleaning can become slow, leading to caches filling up beyond their configured limits. For performance reasons it may be advantageous to disable cache cleaning by the A-REX (by removing the `cachesize` command from the config), and run the *cache-clean* tool independently on the machine hosting the file system. (Please refer to Section **5.1.5**, *Cache administration.*)

Caches can be added to and removed from the configuration as required without affecting any cached data, but after changing the configuration file, the A-REX should be restarted. If a cache is to be removed and all data erased, it is recommended that the cache be put in a *draining* state until all currently running jobs possibly accessing files in this cache have finished. This can be done by putting the word "drain" as the *link_path*:

```
cachedir=path "drain"
```

For more details about the mechanisms of the cache, please refer to Section **6.5**, *Cache.*

4.4.3.1 The Cache Service

The ARC caching system automatically saves to local disk job input files for use with future jobs. The cache is completely internal to the computing element and cannot be accessed or manipulated from the outside. The ARC Cache Service exposes various operations of the cache and can be especially useful in a pilot job model where input data for jobs is not known until the job is running on the worker node.

It is packaged as `nordugrid-arc-cache-service`, and it can be started with its own init script. For more information about the cache service, please visit the NorduGrid wiki:

`http://wiki.nordugrid.org/index.php/Cache_Service`

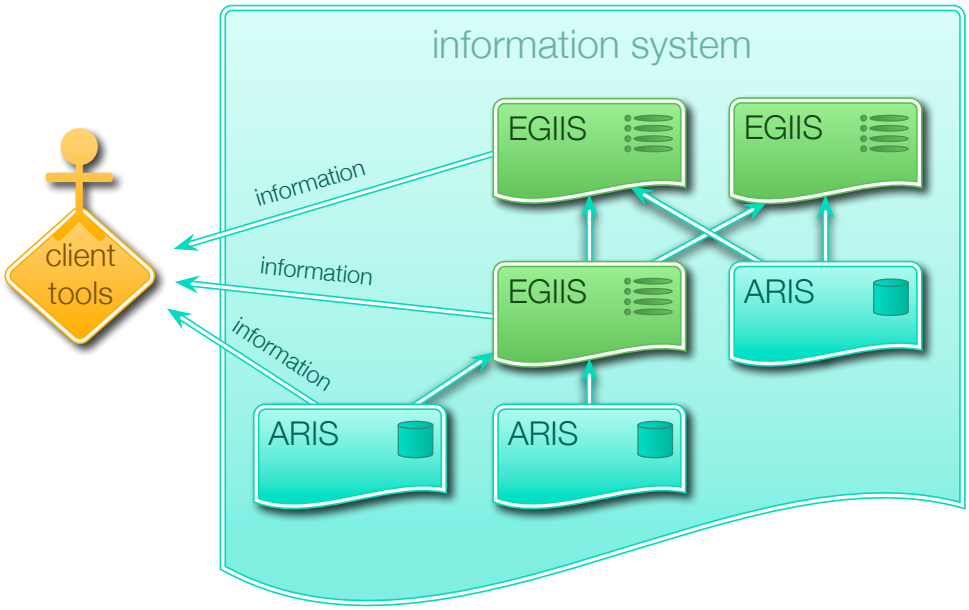


Figure 4.5: The components of the ARC information system: the ARIS which sits next to a computing element (or a storage resource) and advertises information about it; and the EGIIS which indexes the location of ARISes and other EGIIS, creating a hierarchical information mash, where querying the top nodes would provide information about all the resources.

4.4.3.2 The ARC Cache Index (ACIX)

There is another option for locating already cached files: the ARC Cache Index (ACIX). It consists of two components, one on the computing resource: the *Cache server*, and the *Index server* which indexes the cache locations retrieved from the *Cache server*s. This setup allows clients to quickly find the clusters which has the needed input files already cached.

Installation instructions can be found in the `README.txt` files in the NorduGrid subversion:

`http://svn.nordugrid.org/trac/nordugrid/browser/contrib/acix/trunk`

For more information please write a mail to the NorduGrid Discuss mailing list:

`http://mail.nordugrid.org/mailman/listinfo/nordugrid-discuss`

4.4.4 How to join the grid: registering to an index service

Once a cluster is setup, it needs to communicate to some index service to join the grid. Joining an index will let client query the index to find the CE without specifying the CE hostname on the command line.

In the grid world, this is crucial as the user is agnostic about the server his/her jobs will run.

Connection to an index will enable resource sharing in a federated way, among users accepted by the rules in the `[group]` and `[vo]` sections.

NGI usually run their own index, NorduGrid runs several ones:

ldap://index1.nordugrid.org:2135

ldap://index2.nordugrid.org:2135

ldap://index3.nordugrid.org:2135

To connect to an index, add the following to a basic CE configuration file, **after** all the other existing `[infosys]` related sections:

```
...
[infosys/cluster/registration/toPGS1]
targethostname="quark.hep.lu.se"
targetport="2135"
targetsuffix="mds-vo-name=PGS,o=grid"
regperiod="300"
...
```

The special section name `[infosys/cluster/registration/toIndex]` is used to configure registration of a **cluster** (a CE) to an **index service** (an IS[]).

Registration commands explained:

targethostname=*FQDN* – The FQDN of the host running the target index service.

targetport=*portnumber* – Port where the target Index Service is listening. Defaults to 2135.

targetsuffix=*ldapsuffix* – ldap suffix of the target index service. This has to be provided by a manager of the index service, as it is a custom configuration value of the Index Service. Usually is a string of the form `"mds-vo-name=<custom value>,o=grid"`

regperiod=*seconds* – the registration script will be run each number of *seconds*. Defaults to 120.

These commands will affect the way the registration script is run. Logs about registration information can be found by looking at the file configured by the `registrationlog` command in the `[infosys]` section (see Section **4.3.5**, *The [infosys] section: the local information system*). For information on how to read the logs see Section **5.3**, *Log files*

The registration script is called `grid-info-soft-register`. Once registration to an index is configured, parameters of this script can be checked on the system by issuing at the shell:

```
[root@piff tmp]# ps aux | grep reg
root     29718  0.0  0.0  65964  1316 pts/0    S    14:36   0:00
  /bin/sh /usr/share/arc/grid-info-soft-register
  -log /var/log/arc/inforegistration.log
  -f /var/run/arc/infosys/grid-info-resource-register.conf -p 29710

root     29725  0.0  0.0  66088  1320 pts/0    S    14:36   0:00
  /bin/sh /usr/share/arc/grid-info-soft-register
  -log /var/log/arc/inforegistration.log
  -register -t mdsreg2 -h quark.hep.lu.se -p 2135 -period 300
  -dn Mds-Vo-Op-name=register, mds-vo-name=PGS,o=grid -daemon
  -t ldap -h piff.hep.lu.se -p 2135 -ttl 600
  -r nordugrid-cluster-name=piff.hep.lu.se,Mds-Vo-name=local,o=Grid
  -T 45 -b ANONYM-ONLY -z 0 -m cachedump -period 0
```

Other less relevant options are available for registration, please refer to Section **6.1.10**, *Commands in the [infosys/cluster/registration/registrationname] subsections.*

If the registration is successful, the cluster will be shown on the index. To find out that, please refer to the Index Service documentation [].

4.4.5 Accounting with JURA

The A-REX can be configured to periodically execute an external usage reporting utility which should create standard-complient usage records from job usage information provided by the A-REX (called the "job log files") and send the records to remote accounting services. The JURA is such an external utility which capable of doing this. It is distributed with the A-REX.

JURA is capable of creating three types of usage records from the job log files:

- Usage Record 1.0 (UR) XML format [**?**]

- Accounting Processor for Event Logs (APEL) format [**?**]
- Compute Accounting Record (CAR) XML format [**?**]

After creating these usage records, JURA can archive them to a given directory and it can send them to remote services:

- The UR can be sent to an SGAS LUTS (Logging and Usage Tracking Service) [**?**]
- Experimental feature: The APEL usage record can be sent to an APEL service.
- Planned feature: The CAR will be sent to the new version of APEL or any other service supporting the format.

To enable reporting, the `jobreport` and the `jobreport_publisher` configuration commands in the `[grid-manager]` section has to be set to the URL of an accounting destination and the name of the executable publisher. Multiple URLs can be specified in one `jobreport` command, and multiple `jobreport` commands can be used. The usage record of each job will be reported to all of the destinations. Currently if the URL starts with "`CAR`", then a Compute Accounting Record (CAR) will be created, and logged, but it will not be sent anywhere. If a HTTPS URL is given, then then a Usage Record 1.0 will be created and sent to an SGAS LUTS destination. (Currently there is no way to make the A-REX report directly to APEL. The experimental APEL support can be utilized by running JURA separately and specifying a "topic" with the `-t` command line options.) A number can be specified after the URLs: how many days the job log files will be kept if the reporting fails.

The credentials for the HTTPS connection should be set using the `jobreport_credentials` command, specifying first the path to the key then the path to the certificate and the path to the CA certificates directory, separated by space.

Additional options can be given to JURA in the form of comma-separated `key:value` pairs by setting the `jobreport_options` configuration command. Currently these options are recognized:

- **urbatch**:size – JURA sends usage records not one-by-one, but in batches. This options sets the maximum size of a batch. Zero value means unlimited batch size. Default is 50.
- **archiving**:dir – JURA can archive the generated usage records to a given directory. This options specifies the directory and enables archiving. If the directory does not exist, an attempt is made to create it. If this option is absent, no archiving is performed.

An example configuration which will report all jobs to both destination using the given credentials, sending them in batches of 50, and archiving them into `var/urs`:

```
[grid-manager]
jobreport="https://luts1.grid.org:8443/wsrf/services/sgas/LUTS"
jobreport="https://luts2.grid.org:8443/wsrf/services/sgas/LUTS 7"
jobreport_publisher="jura"
jobreport_credentials="/etc/grid-security/hostkey.pem
       /etc/grid-security/hostcert.pem /etc/grid-security/certificates"
jobreport_options="urbatch:50,archiving:/var/urs"
```

For the configuration commands, see also **6.1.11.5**, *Commands related to usage reporting.*

It is also possible to run JURA separately from the A-REX (e.g. a cron job can be set to execute it periodically). The command line options of JURA are the following:

```
jura -E <days> -u <url> -t <topic> -o <path> <control dir>
```

- `-E <days>` – for how many days should failed-to-send records be kept

- `-u <url>` – runs JURA in "interactive mode", which sends usage reports only to the URLs given as command line arguments (and not those which were put into the job log files by the A-REX), and does not delete job log files after a successful report. Multiple `-u` can be given.
- `-t <topic>` – after each `-u <url>` a topic can be specified. This topic is needed for publishing to APEL. If the URL does not start with "CAR" and a topic is specified, the report will be sent to APEL, if a topic is not specified, the report will be sent to SGAS.
- `-o <path>` – specifies the path of the archiving directory, which will be used only for this run of JURA, and the usage records will be put into this directory.
- `<control dir> [<control dir> ...]` – one or more control directories has to be specified. JURA looks for the job log files in the "logs" subdirectory of the control directories given here.

For more details about JURA, see **6.9**, *JURA: The Job Usage Reporter for ARC*.

4.4.6 Sending usage records to SGAS with urlogger

The `urlogger` component of the A-REX is capable to generate and send job usage records to the SGAS [**?**] accounting service.

The following libraries needs to be installed:

- Python 2.4 or later
- Twisted Core and Web (http://twistedmatrix.com/)
- PyOpenSSL (https://launchpad.net/pyopenssl)
- ElementTree (http://effbot.org/zone/element-index.htm - only needed with Python 2.4)

Debian/Ubuntu package names:

`python-twisted-core, python-twisted-web, python-openssl`
`python-elementtree` (only needed with Python 2.4)

RPM based distributions (e.g., RHEL, CentOS, SL, Fedora, etc.):

`python-twisted-core python-twisted-web pyOpenSSL`
`python-elementtree` (only needed with Python 2.4)

The `urlogger` reads its configuration from the `arc.conf`. It should be specified for the A-REX that we want to run the `urlogger` generator script for each job, so the following should be put into the `[grid-manager]` section:

```
[grid-manager]
authplugin="FINISHED timeout=10,onfailure=pass /usr/libexec/arc/arc-ur-logger %C %I %S %U"
```

The plugin will log to the file: `/var/log/arc-ur-logger.log`. This file will not appear until a job has finished.

Then the SGAS service should be specified in a `[logger]` section:

```
[logger]
log_all="https://sgas.ndgf.org:6143/sgas"
```

This will send all records to the given address. It is possible to specify separate SGASes for separate VOs:

```
log_vo="bio.ndgf.org https://biosgas.ndgf.org:6143/sgas"
```

For more options, see Section **6.1.20**, *Commands for the `urlogger` accounting component.*

The A-REX needs to be restarted after the configuration is finished.

A cron job should be set up to send the usage records to SGAS periodically, e.g.:

```
0 * * * * /usr/libexec/arc/arc-ur-registrant
```

To ensure that the registrant is working, you can run the script from the command line first. Note that the script will still write its log to to /var/log/arc-ur-registration.log. By running the script with -s its ouput will be directed to stdout.

4.4.7 Monitoring the ARC CE: Nagios probes

Nagios scripts (probes) exist that allow monitoring of ARC-CEs. The scripts are available in the EGI repository[‡].

NorduGrid provides a set of Nagios tests that can be used to monitor the functionality of an ARC computing element. These tests were originally developed by the NDGF in order to provide availability monitoring to WLCG. The maintenance of the tests has since been taken over by the EMI project.

The tests are available in the workarea of the nordugrid subversion server:

`http://svn.nordugrid.org/trac/workarea/browser/nagios`

They are also available packaged as an RPM: `grid-monitoring-probes-org.ndgf`.

The configuration of the tests is collected in one configuration file called org.ndgf.conf. Adapt it to your needs making sure that the user configured to run the tests is authorized at the CEs you are testing and has the necessary access rights to the storage locations and catalogues configured.

Some of the tests send test jobs to the CE and will report the result when the test job has finished. If the job does not complete within 12 hours it will be killed and a warning is reported in Nagios.

More information about the tests can be found here:

`http://wiki.nordugrid.org/index.php/Nagios_Tests`

4.5 Enhancing CE capabilities

Once a basic CE is in place and its basic functionalities have been tested, is possible to add more features to it.

These include:

Enable glue1.2/1,3, GLUE2 LDAP schemas To be compliant with other grid systems and middlewares, ARC CE can publish its information in these other schemas. In this way its information can show up also in information systems compliant with gLite [**?**]. ARC CE can act as a resource-BDII, to be part of a site-BDII and join the european grid.
See Section **4.5.1**, *Enabling or disabling LDAP schemas*

Enable file caching In grid environments is likely for people to work on the same data set. Downloading an uploading this data can be a time and space consuming task. Hence ARC provides means of caching and hashing data already downloaded and offer it to all the users allowed to run on the cluster, in a completely transparent way.
See Section **4.4.3**, *Enabling the cache*

Provide customized execution environments on-demand As every experiment can have its own libraries, dependencies, tools, ARC provides a mean of creating such environments on demand for each user. This feature is called Runtime Environment (RTE)

Use web services instead/together with of GridFTPd/LDAP Next generation ARC Client and servers are Web Service ready. Job submission and Information System can now be run as a single standardized service using the https protocol, leveraging most of the system administrator's configuration job and increasing performance.

[‡] `https://wiki.egi.eu/wiki/EMI_Nagios_probes`

4.5.1 Enabling or disabling LDAP schemas

ARIS, the cluster information system, can publish information in three schemas and two protocols. Information published via the LDAP protocol can follow the following three schemas:

NorduGrid Schema The default NorduGrid schema, mostly used in Nordic countries and within all the NorduGrid Members. Definition and technical information can be found in [**?**].

Glue 1.2 / 1.3 schema Default currently used by gLite middleware[**?**] and the european grids. Specification can be found here: []

GLUE 2 schema Next generation glue schema with better granularity. Will be the next technology used in production environments. Specification can be found here: [**?**]

The benefits of enabling these schemas are the possibility to join grids different from the nordugrid one, for example to join machines allotted to do special e-Science experiments jobs, such as the ATLAS experiment[**?**].

To enable or disable schema publishing, the first step is to insert the enable commands in the `[infosys]` section as explained in **6.1.5**, *Commands in the [infosys] section.*

Glue 1.2/1.3 schema carries geographycal information and have to be configured in a separate section, **`[infosys/glue12]`**.

If the nordugrid-arc-doc package is installed, two arc.conf examples are available in

`/usr/share/doc/nordugrid-arc-doc/examples/`

Glue 1.2/1.3 `arc_computing_element_glue12.conf`

Glue 2 `arc_computing_element_glue2.conf`

More examples can be found on svn:

`http://svn.nordugrid.org/repos/nordugrid/doc/trunk/examples/`

An example configuration of the [infosys/glue12] section is given in Figure 4.6.

```
[infosys/glue12]
resource_location="Somewhere, Earth"
resource_latitude="54"
resource_longitude="25"
cpu_scaling_reference_si00="2400"
processor_other_description="Cores=1,Benchmark=9.8-HEP-SPEC06"
glue_site_web="http://www.eu-emi.eu"
glue_site_unique_id="MINIMAL Infosys configuration"
provide_glue_site_info="true"
```

Figure 4.6: An example [infosys/glue12] configuration section

Explanation of the commands can be found in the technical reference, section **6.1.7**, *Commands in the [infosys/glue12] section.*

For the GLUE 2.0 it is enough set the command to enable. However, there are other option to let the system administrator configure more features, like the AdminDomain information used for a cluster to join a domain that might be distributed across different geographical sites. A minimal example is detailed in Figure 4.7 and it just contains the domain name. For detailed information please see **6.1.6**, *Commands in the [infosys/admindomain] section.*

```
[infosys/admindomain]
name="ARC-TESTDOMAIN"
```

Figure 4.7: An example [infosys/admindomain] configuration section

4.5.1.1 Applying changes

Once arc.conf is modified, restart the infosystem as explained in Section **5.1**, *Starting and stopping CE services.*

To test information is being published, follow the instructions in Section **5.2.1**, *Testing the information system.*

4.5.2 Runtime Environments

For publicly available runtime environments please see the RTE repository at `http://gridrer.csc.fi/`.

The A-REX can run specially prepared *BASH* scripts prior to creation of the job's script, before and after executing job's main executable. Those scripts are requested by the user through the *runtimeenvironment* attribute in JSDL and are run with the only argument set either equal to '0', '1' or '2' during creation of the job's script, before execution of the main executable and after main the executable is finished, respectively. They all are run through BASH's 'source' command, and hence can manipulate shell variables. With argument '0' scripts are run by the A-REX on the frontend. Some environment variables are defined in that case and can be changed to influence job's execution later:

- joboption_directory – session directory of job.
- joboption_controldir – control directory of job. Various internal information related to this job is stored in file in that directory under names job.*job_gridid*.*. For more information see section **??**.
- joboption_arg_# – command with arguments to be executed as specified in the JD (**not** bash array).
- joboption_arg_code – exit code expected from executable if execution succeeded.
- joboption_pre_#_# – command with arguments to be executed before main executable (**not** bash array). There may be multiple such pre-executables numbered from 0.
- joboption_pre_#_code – exit code expected from corresponding pre-executable if execution succeeded.
- joboption_post_#_# – command with arguments to be executed after main executable (**not** bash array). There may be multiple such post-executables numbered from 0.
- joboption_post_#_code – exit code expected from corresponding post-executable if execution succeeded.
- joboption_stdin – name of file to be attached to stdin handle.
- joboption_stdout – same for stdout.
- joboption_stderr – same for stderr.
- joboption_env_# – array of 'NAME=VALUE' environment variables (**not** bash array).
- joboption_cputime – amount of CPU time requested (minutes).
- joboption_walltime – amount of execution time requested (minutes).
- joboption_memory – amount of memory requested (megabytes).
- joboption_count – number of processors requested.
- joboption_runtime_# – array of requested *runtimeenvironment* names (**not** bash array).
- joboption_num – *runtimeenvironment* currently beeing processed (number starting from 0).

- joboption_jobname – name of the job as given by user.
- joboption_lrms – LRMS to be used to run job.
- joboption_queue – name of a queue of LRMS to put job into.
- joboption_starttime – execution start time as requested in the JD in MDS format.
- joboption_gridid – identifier of the job assigned by A-REX. It is opaque string representing job inside A-REX service. It may be not same as job identifier presented to external client.
- joboption_inputfile_# – local name of pre-staged file (**not** bash array).
- joboption_outputfile_# – local name of file to be post-staged or kept locally after execution (**not** bash array).
- joboption_localtransfer – if set to 'yes' data staging is done on computing node.
- joboption_nodeproperty_# – array of properties of computing nodes (LRMS specific, **not** bash array).

For example *joboption_arg_#* could be changed to wrap the main executable. Or *joboption_runtime* could be expanded if current one depends on others.

With argument '1' scripts are run just before the main executable is run. They are executed on the computing node. Such a script can prepare environment for some third-party software package. A current directory in that case is the one which would be used for execution of the job. Variable `$HOME` also points to that directory.

With argument '2' scripts are executed after main executable finished. Main purpose is to clean possible changes done by scripts run with '1' (like removing temporary files). Execution of scripts at that stage also happens on computing node and is not reliable. If the job is killed by LRMS they most probably won't be executed.

4.5.3 Enabling the Web Services interface

A-REX provides a standard-compliant Web Service (WS) interface to handle job submission/management. The WS interface of A-REX is however disabled by default in ARC and EMI distributions as of 2011. If you are interested to experiment with A-REX advanced features, setting the option `arex_mount_point` in the `[grid-manager]` section of `arc.conf` enables the web service interface, e.g.

```
arex_mount_point="https://your.host:60000/arex"
```

Then you can submit jobs through this new WS interface with the `arcsub` command (available in the ARC client package) and manage jobs with other `arc*` commands.

A-REX also has an EMI Execution Service interface. To enable it, in addition to the above option the following option must be specified

```
enable_emies_interface="yes"
```

IMPORTANT: this web service interface does not accept legacy proxies created by `voms-proxy-init` by default. RFC proxies must be used, which can be created by specifying `voms-proxy-init -rfc` or using `arcproxy`.

The WS interface can run alongside the GridFTP interface. Enabling the WS interface as shown above does not disable the GridFTP interface - if desired "gridftpd" service must be explicitly stopped.

4.5.4 Using Argus authorization service

A-REX with Web Service (WS) interface enabled may use Argus service [**?**] for requesting authorization decision and performing client mapping to local user account. To make A-REX communicate to Argus PEP service for every operation requested through WS interface add following option into the `[grid-manager]` section of `arc.conf`.

```
arguspep_endpoint="https://arguspep.host:8154/authz"
```

A-REX can use different XACML profiles for communicating to Argus PEP. Available are

- direct - pass all authorization attributes (only for debugging). No deployed Argus service implement this profile.
- subject - pass only subject name of client. This is simplified version of 'cream' profile.
- cream - makes A-REX pretend it is gLite CREAM service. This is default and currently recommended profile.
- emi - new profile devloped in EMI project. By the time of writing no known Argus services implement this profile.

```
arguspep_profile="cream"
```

To choose either username of local account provided by Argus PEP should be accepted `arguspep_usermap` option is used. By default local account name provided by Argus is ignored. Setting

```
arguspep_usermap="yes"
```

IMPORTANT: note that first mapping rules defined in `[grid-manager]` section are prcessed and then Argus is contacted. Hence account name provided by Argus will overwrite one defined by local rules.

IMPORTANT: although direct communication with Argus PEP server is only possible for WS enabled server it is possible to use Argus command line utilities as authorization and account mapping plugins in `[gridftpd]` section of configuration file.

```
  [grid-manager]
  authplugin="ACCEPTED timeout=20 pepcli_wrapper.sh %C/job.%I.proxy"

Content of pepcli_warpper.sh:
  #!/bin/sh
  pepcli --pepd https://arguspep.host:8154/authz --certchain "$1" -v --cert /etc/grid-se
```

Example above uses authplugin feature of A-REX to perform authorization for job submission operation. More sophisticated scenarios may be covered by more complex pepci-wrapper.sh. For more information see Argus documentation [**?**] and description of various plugins sections 6.1.3, 6.1.4 and 6.1.11.6.

4.5.5 Virtual Organization Membership Service (VOMS)

Classic authentication of users in grid environment is based on his/her certificate DN. Authorization of users is performed by checking the lists of permitted user DNs, also known as grid-mapfiles. Classic scheme is the simplest to deal with, but it may have scalability and flexibility restrictions when operating with dynamic groups of researchers – Virtual Organizations (VO).

From the computing element perspective, all members of a particular VO are involved in the same research field having common predictable requirements for resources that allows flexibly configure LRMS scheduler policies. In general, VOs have an internal structure that regulate relationships between members that is

implemented via groups, roles and attributes. VO membership parameters are controlled by means of the VOMS specialized software[§].

VOMS consists of two parts:

- VO Management interface (VOMS-Admin) – web-based solution to control membership parameters. Along with the management interface, the service provides SOAP interface to generate lists of VO members' DNs. EDG VOMS-Admin is a classic VO Management solution distributed by EMI [**?**]. There is also alternative lightweight solution available – PHP VOMS-Admin [**?**].
- Credentials signing service (vomsd) – standalone daemon that fortify user VO membership and its parameters. Credentials signing daemon issues an Attribute Certificate (AC) extension attached to user's proxy-certificate and is used in a delegation process. VOMS processing API of the middleware or some external authorization processing executables may parse and verify VOMS AC extension and make a decision taking into account group affiliation instead of just using personal certificate DN.

To maintain the grid-mapfiles based on information in the VOMS database (using SOAP interface of the VO Management service), you need to use `voms://` or `vomss://` sources in [vo] configuration block for the `nordugridmap` utility (see section **6.1.2**, *Commands in the [vo] section* for details).

VOMS credentials signing daemon used directly by client tools (see `arcproxy` manual) to create VOMS AC-enabled proxy. Computing element does not interact with credentials signing daemon directly, but verifies the digital signature of the VOMS server against a configured list of trusted VOMS AC issuers instead.

All VOMS-related configurations described below are supported by ARC API as well as other EMI products based on classic VOMS libraries.

4.5.5.1 Configuring trusted VOMS AC issuers

The VOMS AC signature included in client's proxy certificate can be verified in two ways:

1. Get the issuing VOMS server certificate to trust beforehand and use it for signature verification.
2. Configure the lists of certificates (LSC) to verify certificate chain in the VOMS AC.

Getting the VOMS server certificate. THIS CONFIGURATION METHOD IS OBSOLETE AND NOT SUPPORTED SINCE ARC 1.0.0!

Among all EGI-supported grid services there are only few that do not support LSC files configuration – glite-FTS and glite-WMS for gLite 3.1. If you require legacy VOMS credentials setup for those services, please refer to appropriate documentation.

LSC Files. The trust chain from Certificate Authority (CA) to the VOMS AC issuing server certificate needs to be described in order to verify ACs in clients' proxies issued by that server. Generally, VOMS server certificate is signed by CA directly, so there is only two certificates in the chain of trust, but it can be much longer in other cases.

Each line the LSC file lists a single certificate DN starting from the VOMS server and continues up the trust chain ending with the root CA certificate DN. Following the example of LSC file for voms.ndgf.org server:

```
/O=Grid/O=NorduGrid/CN=host/voms.ndgf.org
/O=Grid/O=NorduGrid/CN=NorduGrid Certification Authority
```

In some rare cases (e.g. host certificate change and/or moving to different CA) it is possible to specify several lists per hostname, separating them with `------ NEXT CHAIN ------` line (ARC parser uses `NEXT CHAIN` match only, but classic VOMS libraries require exactly six dashes and space around it, so you should probably put it there for compatibility).

Following the example of several chains in a single LSC-file:

[§]There are also another technologies exist for group management, but VOMS is the most popular and widely supported

```
/DC=org/DC=ugrid/O=hosts/O=KNU/CN=host/grid.org.ua
/DC=org/DC=ugrid/CN=UGRID CA
------ NEXT CHAIN ------
/DC=org/DC=ugrid/O=hosts/O=KNU/CN=grid.org.ua
/DC=org/DC=ugrid/CN=UGRID CA
```

To get a trust chain DNs for the VOMS server you can contact the VO manager or use `openssl` for known VOMS servers:

```
echo | openssl s_client -connect <server:port> 2>/dev/null \
| openssl x509 -noout -subject -issuer
```

Here `<port>` is typically the standard VOMS-Admin https interface port – 8443. Port of the `vomsd` daemon listed in the `vomses` file can also be used.

The location of LSC files for ARC is fixed and compatible with other EMI software default setup:

```
/etc/grid-security/vomsdir/<VO>/<hostname>.lsc
```

4.5.5.2 Configuring VOMS AC signing servers to contact

Clients rely on VOMSes configuration. VOMSes refer to a list of VOMS servers that are used to manage the supported VOs, more precisely speaking – VOMS AC signing daemons' contact parameters.

An old compatible way of specifying VOMSes is to put all VOs configuration into a single file `/etc/vomses`. Each line should be written in the following format:

```
"alias" "host address" "TCP port" "host certificate DN" "official VO name"
```

It is advised to have *alias* the same as *official VO name*: several VOMS client versions mix them. If several VOMS servers are used by VO for redundancy, you need to specify all of them on the separate lines. You can find this parameters in the "Configuration" section of VOMS-Admin labeled "VOMSES string for this VO".

If you are using recent version of grid software you can maintain a separate VOMSes files for each VO. This files should be placed in VOMSes directory – `/etc/grid-security/vomses/` is used by default but can be redefined with `X509_VOMSES` environmental variable. Please refer to client documentation for more information. For example, to configure support of `nordugrid.org` VO you need to create a file `/etc/grid-security/vomses/nordugrid.org` with the following content:

```
"nordugrid.org" "voms.ndgf.org" "15015" "/O=Grid/O=NorduGrid/CN=host/↩
    voms.ndgf.org" "nordugrid.org"
```

4.5.5.3 Configuring ARC to use VOMS extensions

From the client side, `arcproxy` already has built-in support for VOMS AC extensions, so no additional configuration is required unless you want to redefine VOMSes path.

To utilize VOMS AC extensions in A-REX you have several possibilities:

- using access control filter based on VOMS AC (see section **4.4.1**, *Access control: users, groups, VOs* for details)
- utilizing LCAS/LCMAPS authorization and mapping (see section **4.5.7**, *Using LCAS/LCMAPS* for details)
- using external plugins that operate with VOMS AC (e.g. `arc-vomsac-check`)

4.5.6 Dynamic vs static mapping

There are many debates on using static or dynamic local account mapping policy. Historically, ARC has initially supported only static mapping. Currently, ARC and all middlewares involved into EMI project support and can be configured to use either static or dynamic mapping policy or even some combination of them.

4.5.6.1 Static mapping

The main reason of using static account mapping policy is to *simplify administration of grid services*. Static mapping works by assigning a fixed operating system account to a grid user identified by his/her DN. General practice is mapping all grid users to one or few operating system accounts dedicated to grid jobs.

The most significant drawback is that the different grid users are *indistinguishable* for the underlying system infrastructure. There is no ability to securely isolate different jobs running with the same credentials or implement complex scheduling policy in the LRMS with reservations and priorities as well as employ flexible disk space allocation policy.

In other case, if every grid user is mapped to a dedicated local account, there is significant increase of administration burden. Individual mappings and their permissions should be manually synchronozed with grid user directories (like VOMS or Globus CAS).

Dynamic mapping Dynamic mapping policy allows to provide every grid user with a separate dynamically leased local account and deploy more secure and flexible configurations. Generally, dynamic mapping involves using multiple pools of local accounts for different classes of grid users.

Common examples of such classes include VOs and groups/roles in the VOs. This allows for building authorization and mapping policies in terms of VOMS FQANs additionally to user DNs, which is very common as site usually provide resources for more than one VO.

Each grid user accessing some site service gets mapped to a local account which is leased from an appropriate pool. Policy rules define how to select that pool depending on VOMS AC presented by user as a part of his/her proxy-certificate. User accessing site with different proxies generally will get different maps, depending on FQANs included.

Each grid user gets separated from other local and grid users by means of underlying operating system because with dynamic mapping every grid user is mapped to a dedicated local account. If local account lease is not used for some period of time, it is released and can be assigned to another grid user.

Pool accounts can belong to specific local groups which can be a subject of LRMS scheduling policy or disk quotas. Authorization and mapping policies should be updated only in a case when a new role or group is introduced in a VO, update in case of user membership changes is not necessary.

There are different approaches to implementation of dynamic mapping policy which includes:

- deploying ARC built-in `simplepool` mapping plugin
- using LCMAPS from Site Access Control framework (see section **4.5.7**, *Using LCAS/LCMAPS*)
- using ARGUS dedicated policy service

4.5.7 Using LCAS/LCMAPS

LCAS is the Local Centre Authorization Service. Based on configured policies, LCAS makes a binary authorization decisions. Most of LCAS functionality is covered by ARC internal authorization mechanism (see section **6.1.3**, *Commands in the [group] section*), but it can be used for interoperability to maintain common authorization policy across different flavors of Middleware.

LCMAPS is the Local Credential Mapping Service, it takes care of translating grid credentials to Unix credentials local to the site. LCMAPS (as well as LCAS) is modular and supports flexible configuration of complex mapping policy. This includes not only classical mapping using grid-mapfile generated by `nordugridmap` (see section **4.4.1**, *Access control: users, groups, VOs*) but primarily using dynamic pools and VOMS

AC-based mapping using FQAN match that cannot be accomplished using ARC built-in mechanisms yet. You may consider using LCMAPS to implement VO integration techniques and also for interoperability to maintain common account mapping policy.

LCAS/LCMAPS libraries are provided by Site Access Control (SAC) framework [**?**] that was originally designed to be called from gLite middleware stack and pre-WS part of Globus Toolkit version 4. Nordugrid ARC can also be configured to employ these libraries.

The main goal of using SAC is to maintain common authorization and Unix account mapping policies for a site and employ them on multiple services of a site. The framework allows to configure site-wide autorization policies independently of entry point and consistent mapping among different services that use LCAS/LCMAPS libraries, e.g. A-REX, LCG CE (GT4), CREAM CE or GSISSH.

Additionally, SAC framework provides SCAS mapping service, ARGUS client and gLExec enforcement executable. More information about its functionality and configuration can be found in the SAC documentation [**? ? ?**].

4.5.7.1 Enabling LCAS/LCMAPS

LCAS and LCMAPS can be used via external plugins, which are called out of the GridFTP server. ARC ships with executables `arc-lcas` and `arc-lcmaps` that can be found at `${ARC_LOCATION}/libexec/arc`.

Both executables invoke appropriate functions from shared libraries (`liblcas.so` and `liblcmaps.so` respectively), so you need to have LCAS/LCMAPS installed to use it. Installing SAC framework is not covered by this manual, please refer EMI documentation [**? ?**].

LCAS plugin LCAS plugin is called from `[group]` section that will be used for authentication by `gridftpd` jobplugin or fileplugin. Callout executable requires several parameters:

```
arc-lcas <user DN> <user proxy> <LCAS library name> <LCAS library path> \
             <LCAS policy description file>
```

It can be invoked manually to check the desired operation of LCAS.

User DN and proxy certificate path can be substituted by A-REX using `%D` and `%P` syntax. You need to pass LCAS library name and path to SAC installation location. Syntax of LCAS policy description file is provided later in this section.

Enabling LCAS in `arc.conf` example:

```
[group/users]
plugin="5 /opt/arc/libexec/arc/arc-lcas %D %P liblcas.so /opt/glite/lib /etc/lcas.db"

[gridftpd/jobs]
groupcfg="users"
path="/jobs"
plugin="jobplugin.so"
```

> *NOTE:* There is also another syntax for enabling LCAS in `arc.conf` which can be used in place of `plugin` clause:
>
> ```
> lcas="liblcas.so /opt/glite/lib /etc/lcas.db"
> ```
>
> In older versions of ARC it was used to perform LCAS API callout directly from gridftpd process, but somtimes it resulted in hangs and crashes of the entire process mainly due to incompatibilities between gLite and ARC shared libraries, linked with different globus flavors etc. Now this syntax is left for compatibility and wraps to running the same external executable with 300 seconds timeout.

LCMAPS plugin LCMAPS plugin callout can be set up in the `[gridftpd]` section and can be enabled by `unixmap` or `unixgroup` config options. Callout executable requires several parameters and like LCAS callout, can be invoked manually to verify the LCMAPS configuration itself:

```
arc-lcmaps <user DN> <user proxy> <LCMAPS library name> <LCMAPS library path> \
             <LCMAPS policy description file> <LCMAPS policy name> \
             [<LCMAPS policy name>...]
```

You need to specify all parameters the same way as for the LCAS case. LCMAPS policy description file can define multiple policies, so additional `LCMAPS policy name` parameter is provided to support such a case. Syntax of LCMAPS policy description is provided later in this section.

Enabling LCMAPS in `arc.conf` example:

```
[gridftpd]
gridmap="/dev/null"
allowunknown="yes"
unixmap="* mapplugin 30 /opt/arc/libexec/arc/arc-lcmaps %D %P \
      liblcmaps.so /opt/glite/lib /etc/lcmaps.db voms"
```

Please note: to completely disable checks using classic mapping, you need to specify an empty grid-mapfile in the configuration. As users are no longer present in mapfile, you need to also specify option `allowunknown="yes"`.

> *NOTE:* Another syntax exists for enabling LCMAPS in `arc.conf` the same way as for LCAS:
>
> ```
> unixmap="* lcmaps liblcmaps.so /opt/glite/lib /etc/lcmaps.db voms"
> ```
>
> It wraps to the same executable with 300 secconds timeout for the same reasons as for LCAS.

4.5.7.2 LCAS/LCMAPS policy configuration

LCAS and LCMAPS provide set of plugins to be used for making the policy deceision. All configuration is based on plugins used and their parameters.

LCAS configuration To create an access control policy using LCAS you need to rely on the following a set of basic plugins:

lcas_userallow.mod allows access if DN of the user being checked is listed in the config file provided.

lcas_userban.mod denies access if DN of the user being checked is listed in the config file provided.

lcas_voms.mod checks if FQANs in user's proxy certificate VOMS AC match against config file provided.

lcas_timeslots.mod makes an authorization decisions based on available time slots (as mentioned in LCAS documentation "the most useless plugin ever" :-))

LCAS configuration file (`lcas.db`) contains several lines with the following format:

```
pluginname="<module name/path to plugin file>", pluginargs="<arguments>"
```

Each line represent an authorization policy rule. A positive decision is only reached if all the modules listed permit the user (logical AND).

LCMAPS configuration LCMAPS plugins can belong to one of the two classes, namely acquisition and enforcement. Acquisition modules gather the information about user credentials or find mapping decision that determines user's UID, primary GID and secondary GIDs that can then be assigned by enforcement modules.

LCMAPS basic acquisition modules:

lcmaps_localaccount.mod uses account name corresponding to user's DN in static mapfile (mostly like classic grid-mapfile).

lcmaps_poolaccount.mod allocates account from a pool corresponding to user's DN in static mapfile (like grid-mapfile with "dot-accounts" for Globus with `GRIDMAPDIR` patch).

lcmaps_voms.mod parses and checks proxy-certificate VOMS AC extension and then fills internal LCMAPS data structures with that parsed information, which can be used by other plugins invoked later.

lcmaps_voms_localaccount.mod uses static UID value corresponding to user's VOMS FQAN.

lcmaps_voms_localgroup.mod uses static GID value cerresponding to user's VOMS FQAN.

lcmaps_voms_poolaccount.mod allocates account from a pool corresponding to user's VOMS FQAN.

lcmaps_voms_poolgroup.mod allocate GID from a pool corresponding to user's VOMS FQAN.

lcmaps_scas_client.mod passes request to a SCAS server for making the decision.

LCMAPS basic enforcement modules:

lcmaps_posix_enf.mod sets UID/GID by POSIX `setreuid()/setregid()` calls so that the LCMAPS caller process after successful enforcement continues running with credentials of an account mapped.

lcmaps_ldap_enf.mod change an information about an account in the LDAP database (uidnumber, gidnumber, memberuid, etc.).

lcmaps_dummy_good.mod does not perform enforcing and returns success in case the mapping was found.

LCMAPS configuration file (`lcmaps.db`) is more complex than LCAS one due to flexibility of policies.

```
# define path to plugable modules
path = /path/to/lcmaps/modules
# define actions
<action1 name> = "<module1 name> [<module1 options>]"
<action2 name> = "<module2 name> [<module2 options>]"
...
<actionM name> = "<moduleN name> [<moduleN options>]"
# define policies
<policy1 name>:
<actionX1> -> <action on success> [| <action on fault>]
<actionX2> -> <action on success> [| <action on fault>]
...
<actionXN> -> <action on success> [| <action on fault>]
...
<policyN name>:
<actionY1> -> <action on success> [| <action on fault>]
<actionY2> -> <action on success> [| <action on fault>]
...
<actionYN> -> <action on success> [| <action on fault>]
```

After specifying path to LCMAPS modules, several actions need to be defined. Each action can be either acquisition or enforcement action, depending on specified module used. If module requires parameters, they are specified just after module filename.

Then defined actions are combined into sequences defining the mapping policy. First line after policy name starts the sequence. Module defined by action from left side of the arrow "->" is executed and depending

on execution result (positive or negative) another action gets called. Action sequence ends on enforcement module execution.

To find more information about available pluggable modules and their configuration options, please follow the LCMAPS documentation [**?**].

Environmental variables To fine-tune or debug LCAS/LCMAPS framework operation, special environmental variables should be used. There is no another way to change e.g. debug level.

LCAS environmental variables:

LCAS_LOG_FILE sets location of the logfile

LCAS_LOG_TYPE determines method of logging (`logfile`, `syslog`, `both` or `none`)

LCAS_LOG_STRING specifies text to be prepended to each line to be logged

LCAS_DB_FILE specifies location of lcas policy file (either absolute or relative to `LCAS_DIR`)

LCAS_DEBUG_LEVEL sets debug level (0-5)

LCAS_MOD_DIR sets location of the LCAS plugins (`/modules` will be added to the end of value specified)

LCAS_DIR sets location of LCAS configuration files

LCAS_ETC_DIR can be used alternatively to `LCAS_DIR` for the same purpose

LCMAPS enviromental variables:

LCMAPS_LOG_FILE sets location of the logfile

LCMAPS_LOG_TYPE determines method of logging (`logfile`, `syslog`, `both` or `none`)

LCMAPS_LOG_STRING specifies text to be prepended to each line to be logged

LCMAPS_DB_FILE specifies location of lcas policy file (either absolute or relative to `LCMAPS_DIR`)

LCMAPS_DEBUG_LEVEL sets debug level (0-5)

LCMAPS_MOD_DIR sets location of the LCMAPS plugins (`/modules` will be added to the end of value specified)

LCMAPS_DIR sets location of LCAS configuration files

LCMAPS_ETC_DIR can be used alternatively to `LCMAPS_DIR` for the same purpose

LCMAPS_POLICY_STRING determines the list of policies to apply from a configuration file

4.5.7.3 Example LCAS configuration

Here is an example of LCAS configuration file:

```
pluginname=lcas_userban.mod,pluginargs=/etc/grid-security/lcas/ban_users.db
pluginname=lcas_voms.mod,pluginargs="-vomsdir /etc/grid-security/vomsdir/"
" -certdir /etc/grid-security/certificates/"
" -authfile /etc/grid-security/voms-user-mapfile"
" -authformat simple"
```

There are two modules used: `lcas_userban.mod` and `lcas_voms.mod`. The list of particular users to ban (their certificate DNs) is stored in the file `/etc/grid-security/lcas/ban_users.db` that is passed to `lcas_userban.mod`.

If user's certificate DN is not directly banned, then VO membership check is performed by `lcas_voms.mod`. Plugin accepts several parameters: `vomsdir` and `certdir` paths used to check proxy-certificate and VOMS AC extension; `authfile` contains allowed FQANs specified in format set by `authformat`.

Example content of `/etc/grid-security/voms-user-mapfile`:

```
"/dteam" .dteam
"/dteam/Role=lcgadmin" .sgmdtm"
"/dteam/Role=NULL/Capability=NULL" .dteam
"/dteam/Role=lcgadmin/Capability=NULL" .sgmdtm
"/VO=dteam/GROUP=/dteam" .dteam
"/VO=dteam/GROUP=/dteam/ROLE=lcgadmin" .sgmdtm
"/VO=dteam/GROUP=/dteam/ROLE=NULL/Capability=NULL" .dteam
"/VO=dteam/GROUP=/dteam/ROLE=lcgadmin/Capability=NULL" .sgmdtm
```

Only the first parameter (FQAN) is used. The second parameter is valuable only for LCMAPS, when it is configured to use the same file. The several FQAN specification formats are used to support different versions of VOMS library. If the latest VOMS library (later than version 2.0.2 from EMI-1) is installed on a site then just the first two lines are enough, but to keep things safe and support older VOMS, all of them should be given.

GACL format of `authfile` can also be used as well as more options and plugins. Please refer LCAS documentation for more information.

4.5.7.4 Example LCMAPS configuration

LCMAPS configuration for ARC is not an enforcing configuration (it means that LCMAPS does not actually apply UID/GID assignment on execution), so we need to use `lcmaps_dummy_good.mod` plugin to accomplish this. `arc-lcmaps` executable returns user name and optionally group name to stdout which is then used by ARC to perform enforcing by itself.

Simple gridmap behavior For gridmap behaviour you can just use `lcmaps_localaccount.mod` plugin with grid-mapfile, where the users are mapped to some Unix account(s).

Example `lcmaps.db` configuration file:

```
path = /opt/glite/lib/modules
# ACTIONS
# do not perform enforcement
good = "lcmaps_dummy_good.mod"
# statically mapped accounts
localaccount = "lcmaps_localaccount.mod"
" -gridmapfile /etc/grid-security/grid-mapfile"

# POLICIES
staticmap:
localaccount -> good
```

There is only one policy `staticmap` defined: after `localaccount` action is called, LCMAPS execution gets finished.

VOMS AC-based mapping to pools Parsing VOMS AC is accomplished via `lcmaps_voms` family of plugins. Account pools and `gridmapdir` should be created beforehand.

```
path = /opt/glite/lib/modules
# ACTIONS
# do not perform enforcement
good = "lcmaps_dummy_good.mod"
# parse VOMS AC to LCMAPS data structures
vomsextract = "lcmaps_voms.mod"
" -vomsdir /etc/grid-security/vomsdir"
" -certdir /etc/grid-security/certificates"
# FQAN-based pool account mapping
vomspoolaccount = "lcmaps_voms_poolaccount.mod"
" -override_inconsistency"
```

```
" -max_mappings_per_credential 1"
" -do_not_use_secondary_gids"
" -gridmapfile /etc/grid-security/voms-user-mapfile"
" -gridmapdir /etc/grid-security/gridmapdir"
# FQAN-based group mapping
vomslocalgroup = "lcmaps_voms_localgroup.mod"
" -groupmapfile /etc/grid-security/voms-group-mapfile"
" -mapmin 1"

#POLICIES
voms:
vomsextract -> vomspoolaccount | good
vomspoolaccount -> vomslocalgroup | good
vomslocalgroup -> good
```

Configuration requires `voms-group-mapfile` which maps FQANs to groups and `voms-user-mapfile` which maps FQANs to accounts from pools. Directories `vomsdir` and `certdir` in `vomsextract` configuration are used to check VOMS AC validity.

Example content of `/etc/grid-security/voms-user-mapfile` is provided in section **4.5.7.3**, *Example LCAS configuration*. Second parameter indicate Unix account used to accomplish mapping for specified FQAN. Notice the dot prepending an account name – that meens that free pool account will be used instead of a single account.

For example, there are 50 pool accounts named `dteam01`, `dteam02` ...`dteam50`. Specifying `.dteam` in `voms-user-mapfile` means that LCMAPS need to get any unused account from the pool and assign it to user's DN:FQAN pair. Already leased accounts are tracked by hard-linking account file to url-encoded DN:FQAN pair file in the `gridmapdir`.

File `voms-group-mapfile` is simillar to `voms-user-mapfile`, but the second parameter indicates a group name. If parameter has a dot prepended like in `voms-user-mapfile`, it defines the name of the pool of groups that can be used with `lcmaps_voms_poolgroup.mod` plugin.

There is the only one policy defined – `voms`. Action `vomsextract` gets executed first and on success `vomspoolaccount` module is used. If validation and parsing the VOMS AC has failed then LCMAPS execution finishes. When called `vomspoolaccount` allocates account from pool, LCMAPS moves on to finding an appropriate group by `vomslocalgroup` action. LCMAPS execution is finished if `vomspoolaccount` had no success and after `vomslocalgroup` has finished operation successfully.

Chapter 5

Operations

5.1 Starting and stopping CE services

5.1.1 Overview

The scripts needed for a production level CE to work are three:

- `gridftpd` : Starts the gridftpd interface. Brings up the server (configured in the [gridftpd] block) and all the services related to it (configured in all the `[gridftpd/subsection]` blocks).
  Usually located in `/etc/init.d/`
  See Section **4.3.4**, *The [gridftpd] section: the job submission interface* for configuration details.
- `a-rex` : Starts A-REX, the grid manager (configured in the `[grid-manager]` block). It prepares the configfiles and starts the `arched` hosting environment process.
  Starts the Web Services interface **only if** it has been enabled. See Section **4.5.3**, *Enabling the Web Services interface*
  Starts LRMS scripts. See Section **4.4.2**, *Connecting to the LRMS* for configuration details.
  Usually located in `/etc/init.d/`
  See Section **4.3.3**, *The [grid-manager] section: setting up the A-REX and the arched* for configuration details.
- `grid-infosys` : Starts the LDAP server and the infosystem scripts (configured in the `[infosys]` configuration block and its subsections).
  Usually located in `/etc/init.d/`
  See Section **4.3.5**, *The [infosys] section: the local information system* for configuration details.

5.1.2 Starting the CE

To start a CE, issue the following commands with root rights in the following order:

1. `# service gridftpd start`
2. `# service a-rex start`
3. `# service grid-infosys start`

Alternatively the exact same procedure can be used calling the scripts directly:

1. `# /etc/init.d/gridftpd start`
2. `# /etc/init.d/service a-rex start`
3. `# /etc/init.d/service grid-infosys start`

5.1.3 Stopping the CE

To stop a CE, issue the following commands with root rights in the following order:

1. `# service grid-infosys stop`
2. `# service a-rex stop`
3. `# service gridftpd stop`

Alternatively the exact same procedure can be used calling the scripts directly:

1. `# /etc/init.d/service grid-infosys stop`
2. `# /etc/init.d/service a-rex stop`
3. `# /etc/init.d/gridftpd stop`

5.1.4 Verifying the status of a service

To check the status of a service, issue the command:

```
# service <servicename> status
```

Alternatively the exact same procedure can be used calling the scripts directly:

```
# /etc/init.d/<servicename> status
```

where `<servicename>` is one of `gridftpd`, `a-rex`, `grid-infosys`

Depending on the security configuration, root permissions might be needed to execute these commands.

A CE is fully funtional when all the three scripts return and OK status with a process PID of each service process.

5.1.5 Cache administration

The following tools exist to help with administration of the cache:

- *cache-clean* - This tool is used periodically by the A-REX to keep the size of each cache within the configured limits.
  cache-clean -h gives a list of options. The most useful option for administrators is *-s*, which does not delete anything, but gives summary information on the files in the cache, including information on the ages of the files in the cache.
  It is not recommended to run *cache-clean* manually to clean up the cache, unless it is desired to temporarily clean up the cache with different size limits to those specified in the configuration, or to improve performance by running it on the file system's local node as mentioned above.
- *cache-list* - This tool is used to list all files present in each cache or, given a list of URLs as arguments, shows the location of each URL in the cache if present. In the first case it simply reads through all the *.meta* files and prints to stdout a list of all URLs stored in each cache and their corresponding cache filename, one per line. In the second case the cache filename of each URL is calculated and then each cache is checked for the existence of the file.

5.2 Testing a configuration

This chapter will give instructions on how to test and troubleshoot that a given configuration is correct, and that everything is running properly.

Things to check are, in order of importance:

1. **The information system is running and publishing the correct information.** Without a properly configured information systems, the clients will not be able to query the cluster for its resources and do an efficient brokering.
   See Section **5.2.1**, *Testing the information system*
2. **A-REX is running with valid certificates installed**.
   See Section **5.2.2**, *Testing whether the certificates are valid*
3. **The job submission interface is listening and accepting for jobs**.
   See Section **5.2.3**, *Testing the job submission interface*
4. **LRMS configuration is correct and a job can be executed on the queues**.
   See Section **5.2.4**, *Testing the LRMS*

5.2.1 Testing the information system

ARC-CE information system publishes in LDAP and WebServices/XML format.

To test if the LDAP information system is running, ldap tools must be installed. In particular the tool called `ldapsearch` [**?**].

To test if the WS information system is running, ARC suggests its own tool called `arcwsrf` [**?**].

5.2.1.1 Check NorduGrid Schema publishing

To check if the information system is creating the needed ldap trees and publishing them, issue the following:

```
ldapsearch -x -H ldap://localhost:2135 -b 'mds-vo-name=local,o=grid'
```

and the result should be something like the one in Figure 5.1.

To check that the information system is publishing **outside** the cluster, i.e. on its public IP, execute the same query on its hostname, preferrably from a remote machine:

```
ldapsearch -x -H ldap://<hostname>:2135 -b 'mds-vo-name=local,o=grid'
```

The result must be the similar to the one in Figure 5.1.

All the values must be consistent with you setup. For example, nordugrid-cluster-name must be the machine's hostname.

5.2.1.2 Check Glue 1.x Schema publishing

To check if the information system is creating the needed ldap trees and publishing them, issue the following:

```
ldapsearch -x -H ldap://localhost:2135 -b 'mds-vo-name=resource,o=grid'
```

and the result should be something like the one in Figure 5.2.

To check that the information system is publishing **outside** the cluster, i.e. on its public IP, execute the same query on its hostname, preferrably from a remote machine:

```
ldapsearch -x -H ldap://<hostname>:2135 -b 'mds-vo-name=resource,o=grid'
```

The result must be the similar to the one in Figure 5.2.

```
 # extended LDIF
#
# LDAPv3
# base <mds-vo-name=local,o=grid> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#

# local, Grid
dn: Mds-Vo-name=local,o=Grid
objectClass: Mds
objectClass: MdsVo
Mds-Vo-name: local
Mds-validfrom: 20110811172014Z
Mds-validto: 201108111820l4Z

# piff.hep.lu.se, local, grid
dn: nordugrid-cluster-name=piff.hep.lu.se,Mds-Vo-name=local,o=grid
nordugrid-cluster-totalcpus: 2
nordugrid-cluster-homogeneity: TRUE
nordugrid-cluster-name: piff.hep.lu.se
nordugrid-cluster-lrms-version: 0.9
nordugrid-cluster-middleware: nordugrid-arc-1.0.1
nordugrid-cluster-middleware: globus-5.0.3
nordugrid-cluster-trustedca: /O=Grid/O=NorduGrid/CN=NorduGrid Certification Au
 thority
nordugrid-cluster-cpudistribution: 2cpu:1
nordugrid-cluster-sessiondir-lifetime: 10080
nordugrid-cluster-issuerca: /DC=eu/DC=KnowARC/CN=LUEMI-1310134495.12
nordugrid-cluster-credentialexpirationtime: 20110807141455Z
nordugrid-cluster-lrms-type: fork
nordugrid-cluster-sessiondir-free: 129566
nordugrid-cluster-sessiondir-total: 143858
nordugrid-cluster-architecture: x86_64
nordugrid-cluster-prelrmsqueued: 0
nordugrid-cluster-comment: This is a minimal out-of-box CE setup
nordugrid-cluster-contactstring: gsiftp://piff.hep.lu.se:2811/jobs
nordugrid-cluster-issuerca-hash: 8050ebf5
nordugrid-cluster-totaljobs: 0
nordugrid-cluster-aliasname: MINIMAL Computing Element
nordugrid-cluster-usedcpus: 0
objectClass: Mds
objectClass: nordugrid-cluster
Mds-validfrom: 20110811172104Z
Mds-validto: 201108111722O4Z

# fork, piff.hep.lu.se, local, grid
dn: nordugrid-queue-name=fork,nordugrid-cluster-name=piff.hep.lu.se,Mds-Vo-nam
 e=local,o=grid
nordugrid-queue-running: 0
```

Figure 5.1: Output of an ldapsearch on a CE

```
 ldapsearch -x -h piff.hep.lu.se -p 2135 -b 'mds-vo-name=resource,o=grid'
# extended LDIF
#
# LDAPv3
# base <mds-vo-name=resource,o=grid> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#

# resource, Grid
dn: Mds-Vo-name=resource,o=Grid
objectClass: Mds
objectClass: MdsVo
Mds-Vo-name: resource
Mds-validfrom: 20110822130627Z
Mds-validto: 20110822140627Z

# piff.hep.lu.se, resource, grid
dn: GlueClusterUniqueID=piff.hep.lu.se,Mds-Vo-name=resource,o=grid
objectClass: GlueClusterTop
objectClass: GlueCluster
objectClass: GlueSchemaVersion
objectClass: GlueInformationService
objectClass: GlueKey
GlueClusterUniqueID: piff.hep.lu.se
GlueClusterService: piff.hep.lu.se
GlueSchemaVersionMinor: 2
GlueForeignKey: GlueCEUniqueID=piff.hep.lu.se:2811/nordugrid-fork-arc
GlueForeignKey: GlueSiteUniqueID=MINIMAL Infosys configuration
GlueSchemaVersionMajor: 1
GlueClusterName: MINIMAL Infosys configuration

# MINIMAL Infosys configuration, resource, grid
dn: GlueSiteUniqueID=MINIMAL Infosys configuration,Mds-Vo-name=resource,o=grid
GlueSiteDescription: ARC-This is a minimal out-of-box CE setup
GlueSiteSecurityContact: mailto: -1
objectClass: GlueTop
objectClass: GlueSite
objectClass: GlueKey
objectClass: GlueSchemaVersion
GlueSiteSysAdminContact: mailto: -1
GlueSiteName: MINIMAL Infosys configuration
GlueSiteUniqueID: MINIMAL Infosys configuration
GlueSchemaVersionMinor: 2
GlueSiteLongitude: 25
GlueSiteLatitude: 54
GlueSchemaVersionMajor: 1
GlueForeignKey: None
GlueSiteOtherInfo: Middleware=ARC
GlueSiteUserSupportContact: mailto: -1
GlueSiteWeb: http://www.eu-emi.eu
GlueSiteLocation: Somewhere, Earth

# piff.hep.lu.se:2811/nordugrid-fork-arc, resource, grid
dn: GlueCEUniqueID=piff.hep.lu.se:2811/nordugrid-fork-arc,Mds-Vo-name=resource
 ,o=grid
GlueCEStateStatus: Production
GlueCEStateTotalJobs: 0
GlueCEInfoJobManager: arc
GlueCEInfoHostName: piff.hep.lu.se
GlueCEUniqueID: piff.hep.lu.se:2811/nordugrid-fork-arc
GlueCEStateFreeJobSlots: 2
GlueForeignKey: GlueClusterUniqueID=piff.hep.lu.se

   ...

# search result
search: 2
result: 0 Success

# numResponses: 9
# numEntries: 8
```

Figure 5.2: Sample glue 1.x infosystem output on a ldap query. The output has been shortened with for ease of reading.

5.2.1.3 Check LDAP GLUE2 Schema publishing

To check if the information system is creating the needed ldap trees and publishing them, issue the following:

```
ldapsearch -x -H ldap://localhost:2135 -b 'o=glue'
```

and the result should be something like the one in Figure 5.3.

To check that the information system is publishing **outside** the cluster, i.e. on its public IP, execute the same query on its hostname, preferrably from a remote machine:

```
ldapsearch -x -H ldap://<hostname>:2135 -b 'o=glue'
```

The result must be the similar to the one in Figure 5.3.

5.2.1.4 Check WS/XML GLUE2 Schema publishing

First you will need to generate a proxy certificate and your grid ID must be allowed on the CE to test, see [].

Call the `arcwsrf` test tool:

```
$ arcwsrf https://<hostname>:<a-rex port>/<a-rex path>
```

where `<a-rex port>` `<a-rex path>` are those specified in Section **4.5.3**, *Enabling the Web Services interface*.

The output should look like in Figure 5.4

5.2.1.5 Further testing hints

If nothing is published or the query hangs, then there can be something wrong with ldap or A-REX.

Check `slapd` logs to find out the problem in the former case, A-REX logs in the latter. Please see also Section **5.3**, *Log files*.

5.2.2 Testing whether the certificates are valid

While A-REX is running, check the logfile specified with the `logfile` option in the `[grid-infosys]` block in `/etc/arc.conf`:

```
[grid-infosys]
 ...
 logfile="/tmp/grid-manager.log"
 ...
```

It will contain information on expired certificates or certificates about to expire, see Figure 5.5.

While ARIS is running, is possible to get that information as well from its logfiles specified with the `providerlog` option in the `[infosys]` block in `/etc/arc.conf` :

```
[infosys]
...
providerlog="/tmp/infoprovider.log"
...
```

It will contain information about expired certificates, see Figure 5.6.

You can inspect the certificates dates by using openssl commands. Please refer to the certificate mini How-to

To understand how to read the logs please refer to Section **5.3**, *Log files*

```
$ ldapsearch -x -h piff.hep.lu.se -p 2135 -b 'o=glue'
[...]
# glue
dn: o=glue
objectClass: top
objectClass: organization
o: glue

# urn:ogf:AdminDomain:hep.lu.se, glue
dn: GLUE2DomainID=urn:ogf:AdminDomain:hep.lu.se,o=glue
objectClass: GLUE2Domain
objectClass: GLUE2AdminDomain
GLUE2EntityName: hep.lu.se
GLUE2DomainID: urn:ogf:AdminDomain:hep.lu.se

# urn:ogf:ComputingService:hep.lu.se:piff, urn:ogf:AdminDomain:hep.lu.se, glue
dn: GLUE2ServiceID=urn:ogf:ComputingService:hep.lu.se:piff,
    GLUE2DomainID=urn:ogf:AdminDomain:hep.lu.se,o=glue
GLUE2ComputingServiceSuspendedJobs: 0
GLUE2EntityValidity: 60
GLUE2ServiceType: org.nordugrid.execution.arex
GLUE2ServiceID: urn:ogf:ComputingService:hep.lu.se:piff
objectClass: GLUE2Service
objectClass: GLUE2ComputingService
GLUE2ComputingServicePreLRMSWaitingJobs: 0
GLUE2ServiceQualityLevel: development
GLUE2ComputingServiceWaitingJobs: 0
GLUE2ServiceComplexity: endpoint=1,share=1,resource=1
GLUE2ComputingServiceTotalJobs: 0
GLUE2ServiceCapability: executionmanagement.jobexecution
GLUE2ComputingServiceRunningJobs: 0
GLUE2ComputingServiceStagingJobs: 0
GLUE2EntityName: piff
GLUE2ServiceAdminDomainForeignKey: urn:ogf:AdminDomain:hep.lu.se
GLUE2EntityCreationTime: 2011-08-22T13:23:24Z

# urn:ogf:ComputingEndpoint:piff.hep.lu.se:443,
  urn:ogf:ComputingService:hep.lu.se:piff, urn:ogf:AdminDomain:hep.lu.se, glue
dn: GLUE2EndpointID=urn:ogf:ComputingEndpoint:piff.hep.lu.se:443,
  GLUE2ServiceID=urn:ogf:ComputingService:hep.lu.se:piff,
  GLUE2DomainID=urn:ogf:AdminDomain:hep.lu.se,o=glue
GLUE2ComputingEndpointRunningJobs: 0
GLUE2ComputingEndpointStaging: staginginout
GLUE2EntityValidity: 60
GLUE2EndpointQualityLevel: development
GLUE2EndpointImplementor: NorduGrid
GLUE2EntityOtherInfo: MiddlewareName=EMI
GLUE2EntityOtherInfo: MiddlewareVersion=1.1.2-1
GLUE2EndpointCapability: executionmanagement.jobexecution
GLUE2EndpointHealthState: ok
GLUE2EndpointServiceForeignKey: urn:ogf:ComputingService:hep.lu.se:piff
GLUE2EndpointTechnology: webservice
GLUE2EndpointWSDL: https://piff.hep.lu.se/arex/?wsdl
GLUE2EndpointInterfaceName: ogf.bes
GLUE2ComputingEndpointWaitingJobs: 0
GLUE2ComputingEndpointComputingServiceForeignKey: urn:ogf:ComputingService:hep.lu.se:piff
GLUE2EndpointURL: https://piff.hep.lu.se/arex
GLUE2ComputingEndpointSuspendedJobs: 0
GLUE2EndpointImplementationVersion: 1.0.1
GLUE2EndpointSemantics: http://www.nordugrid.org/documents/arex.pdf
GLUE2ComputingEndpointPreLRMSWaitingJobs: 0
GLUE2EndpointIssuerCA: /DC=eu/DC=KnowARC/CN=LUEMI-1313588355.29
GLUE2EndpointServingState: production
GLUE2ComputingEndpointStagingJobs: 0
objectClass: GLUE2Endpoint
objectClass: GLUE2ComputingEndpoint
GLUE2EndpointInterfaceVersion: 1.0
GLUE2EndpointSupportedProfile: http://www.ws-i.org/Profiles/BasicProfile-1.0.html
GLUE2EndpointSupportedProfile: http://schemas.ogf.org/hpcp/2007/01/bp
GLUE2EndpointImplementationName: ARC
GLUE2EndpointTrustedCA: /DC=eu/DC=KnowARC/CN=LUEMI-1313588355.29
GLUE2EndpointTrustedCA: /O=Grid/O=NorduGrid/CN=NorduGrid Certification Authority
GLUE2ComputingEndpointJobDescription: ogf:jsdl:1.0
GLUE2ComputingEndpointJobDescription: nordugrid:xrsl
GLUE2EndpointID: urn:ogf:ComputingEndpoint:piff.hep.lu.se:443
GLUE2EntityCreationTime: 2011-08-22T13:23:24Z
[...]
# search result
search: 2
result: 0 Success
# numResponses: 6
# numEntries: 5
```

Figure 5.3: Sample LDAP search output on GLUE2 enabled infosystem. The output has been shortened with [...] for ease of reading.

```
<wsrf-rp:GetResourcePropertyDocumentResponse><InfoRoot>
  <Domains xmlns="http://schemas.ogf.org/glue/2008/05/spec_2.0_d41_r01" [...]>
    <AdminDomain BaseType="Domain">
      <ID>urn:ogf:AdminDomain:hep.lu.se</ID>
      <Name>hep.lu.se</Name>
      <Services>
        <ComputingService BaseType="Service" CreationTime="2011-08-22T13:34:56Z" Validity="60">
          <ID>urn:ogf:ComputingService:hep.lu.se:piff</ID>
          <Name>piff</Name>
          <Capability>executionmanagement.jobexecution</Capability>
          <Type>org.nordugrid.execution.arex</Type>
          <QualityLevel>development</QualityLevel>
          <Complexity>endpoint=1,share=1,resource=1</Complexity>
          <TotalJobs>0</TotalJobs>
          <RunningJobs>0</RunningJobs>
          <WaitingJobs>0</WaitingJobs>
          <StagingJobs>0</StagingJobs>
          <SuspendedJobs>0</SuspendedJobs>
          <PreLRMSWaitingJobs>0</PreLRMSWaitingJobs>
          <ComputingEndpoint BaseType="Endpoint" CreationTime="2011-08-22T13:34:56Z" Validity="60">
            <ID>urn:ogf:ComputingEndpoint:piff.hep.lu.se:60000</ID>
            <OtherInfo>MiddlewareName=EMI</OtherInfo>
            <OtherInfo>MiddlewareVersion=1.1.2-1</OtherInfo>
            <URL>https://piff.hep.lu.se:60000/arex</URL>
            <Capability>executionmanagement.jobexecution</Capability>
            <Technology>webservice</Technology>
            <InterfaceName>ogf.bes</InterfaceName>
            <InterfaceVersion>1.0</InterfaceVersion>
            <WSDL>https://piff.hep.lu.se:60000/arex/?wsdl</WSDL>
            <SupportedProfile>http://www.ws-i.org/Profiles/BasicProfile-1.0.html</SupportedProfile>
            <SupportedProfile>http://schemas.ogf.org/hpcp/2007/01/bp</SupportedProfile>
            <Semantics>http://www.nordugrid.org/documents/arex.pdf</Semantics>
            <Implementor>NorduGrid</Implementor>
            <ImplementationName>ARC</ImplementationName>
            <ImplementationVersion>1.0.1</ImplementationVersion>
            <QualityLevel>development</QualityLevel>
            <HealthState>ok</HealthState>
            <ServingState>production</ServingState>
            <IssuerCA>/DC=eu/DC=KnowARC/CN=LUEMI-1313588355.29</IssuerCA>
            <TrustedCA>/DC=eu/DC=KnowARC/CN=LUEMI-1313588355.29</TrustedCA>
            <TrustedCA>/O=Grid/O=NorduGrid/CN=NorduGrid Certification Authority</TrustedCA>
            <Staging>stagininout</Staging>
            <JobDescription>ogf:jsdl:1.0</JobDescription>
            <JobDescription>nordugrid:xrsl</JobDescription>
            <TotalJobs>0</TotalJobs>
            <RunningJobs>0</RunningJobs>
            <WaitingJobs>0</WaitingJobs>
            <StagingJobs>0</StagingJobs>
            <SuspendedJobs>0</SuspendedJobs>
            <PreLRMSWaitingJobs>0</PreLRMSWaitingJobs>
            <Associations>
              <ComputingShareID>urn:ogf:ComputingShare:hep.lu.se:piff:fork</ComputingShareID>
            </Associations>
            <ComputingActivities>
            </ComputingActivities>
          </ComputingEndpoint>
          <ComputingShare BaseType="Share" CreationTime="2011-08-22T13:34:56Z" Validity="60">
            <ID>urn:ogf:ComputingShare:hep.lu.se:piff:fork</ID>
            <Name>fork</Name>
            <Description>This queue is nothing more than a fork host</Description>
            <MappingQueue>fork</MappingQueue>
[...]
            <PreLRMSWaitingJobs>0</PreLRMSWaitingJobs>
            <FreeSlots>2</FreeSlots>
            <FreeSlotsWithDuration>2</FreeSlotsWithDuration>
            <UsedSlots>0</UsedSlots>
            <RequestedSlots>0</RequestedSlots>
            <Associations>
              <ComputingEndpointID>urn:ogf:ComputingEndpoint:piff.hep.lu.se:60000</ComputingEndpointID>
              <ExecutionEnvironmentID>urn:ogf:ExecutionEnvironment:hep.lu.se:piff:fork</ExecutionEnvironmentID>
            </Associations>
          </ComputingShare>
          <ComputingManager BaseType="Manager" CreationTime="2011-08-22T13:34:56Z" Validity="60">
            <ID>urn:ogf:ComputingManager:hep.lu.se:piff</ID>
[...]
          </ComputingManager>
        </ComputingService>
      </Services>
    </AdminDomain>
  </Domains>
</InfoRoot>
</wsrf-rp:GetResourcePropertyDocumentResponse>
```

Figure 5.4: Sample ARC WS information system XML output. The output has been shortened with [...] for ease of reading.

```
...
[2011-08-05 11:12:53] [Arc] [WARNING] [3743/406154336] Certificate /DC=eu/DC=KnowARC/CN=LUEMI-1310134495.12
  will expire in 2 days 5 hours 2 minutes 1 second
[2011-08-05 11:12:53] [Arc] [WARNING] [3743/406154336] Certificate /DC=eu/DC=KnowARC/O=Lund University/CN=demo1
  will expire in 2 days 5 hours 2 minutes 1 second
...
```

Figure 5.5: A sample certificate information taken from A-REX logs.

```
...
[2011-08-12 10:39:46] HostInfo: WARNING: Host certificate is expired in file: /etc/grid-security/hostcert.pem
[2011-08-12 10:39:46] HostInfo: WARNING: Certificate is expired for CA: /DC=eu/DC=KnowARC/CN=LUEMI-1305883423.79
[2011-08-12 10:39:46] HostInfo: WARNING: Certificate is expired for CA: /DC=eu/DC=KnowARC/CN=LUEMI-1301496779.44
[2011-08-12 10:39:46] HostInfo: WARNING: Issuer CA certificate is expired in file:
                                         /etc/grid-security/certificates/8050ebf5.0
[2011-08-12 10:39:46] HostInfo: WARNING: Certificate is expired for CA: /DC=eu/DC=KnowARC/CN=LUEMI-1310134495.12
[2011-08-12 10:39:46] HostInfo: WARNING: Issuer CA certificate is expired in file:
                                         /etc/grid-security/certificates/917bb2c0.0
[2011-08-12 10:39:46] HostInfo: WARNING: Certificate is expired for CA: /DC=eu/DC=KnowARC/CN=LUEMI-1310134495.12
[2011-08-12 10:39:46] HostInfo: WARNING: Certificate is expired for CA: /DC=eu/DC=KnowARC/CN=LUEMI-1305883423.79
[2011-08-12 10:39:46] HostInfo: WARNING: Certificate is expired for CA: /DC=eu/DC=KnowARC/CN=LUEMI-1301496779.44
...
```

Figure 5.6: A sample certificate information taken from ARIS logs.

5.2.3 Testing the job submission interface

To test the job submission interface an ARC Client is needed, such as the ng* and arc* tools.

To install an ARC Client refer to `http://www.nordugrid.org/documents/arc-client-install.html`.

Once the clients are installed, the **`arctest`** utility can be used to submit test jobs.

Usage of this tool is out of the scope of this manual. Refer to [**?**] for further information.

To test basic job submission try the following command:

```
arctest -c <hostname fqdn> -J 1
```

The job should at least be submitted succesfully.

5.2.4 Testing the LRMS

Each LRMS has his own special setup. Nevertheless is good practice to follow this approach:

1. submit a job that includes at least these two lines:

   ```
   ("stderr" = "stderr" )
   ("gmlog" = "gmlog" )
   ```

   The first one will pipe all standard errors to a file called stderr, while the second will generate all the needed debugging information in a folder called `gmlog`.

2. retrieve the job with `arcget -a`.

3. In the job session folder just downloaded, check the `gmlog/errors` file to see what the job submission script was and if there are some LRMS related errors.

THe rest of LRMS troubleshooting is LRMS dependent, so please refer to each LRMS specific guide and logs.

5.3 Log files

ARC CE log files paths are configured in `arc.conf` for each component according to the following table:

Component	Configuration section	More information
A-REX	`[grid-manager]`	in subsection 4.3.3
gridftpd interface	`[gridftpd]`	in subsection 4.3.4
infoproviders	`[infosys]`	in subsection 4.3.5
infoproviders ldap server	`[infosys]`	in subsection 4.3.5

5.3.1 The format of the log files

The format of arc log files is the following:

A-REX	[Date]	[Component name]	[error level]	[pid/?]	Message
gridftpd	[Date]	[Component name]	[error level]	[pid/?]	Message
infoproviders	[Date]	infprovider script name:	error level:	Message	
infoprovider registration	Date	pid file of script process	Message		

5.4 Modules of the A-REX

The A-REX consists of several separate modules. These are:

- *libarex.so* – The main module providing main functionality and web interface. It is implemented as HTTP and SOAP service inside HED. It is responsible for processing jobs, moving them through states and running other modules.
- *downloader* – This is a module responsible for gathering input files in the SD. It processes the *job.ID.input* file and updates it.
- *uploader* – This module is responsible for delivering output files to the specified SEs and registration at an Indexing Service (like RLS) as needed. It processes and updates the *job.ID.output* file.
- *gm-kick* – Sends a signal to the A-REX though a FIFO file to wake it up. It's used to increase responsiveness of A-REX.
- *CEinfo.pl* – Collects and generates information about computing resource as XML document in NorduGrid and Glue 2 format.

The following modules are always run under the Unix account to which a Grid user is mapped.

- *smtp-send.sh* and *smtp-send* – These are the modules responsible for sending e-mail notifications to the user. The format of the mail messages can be easily changed by editing the simple shell script *smtp-send.sh*.
- *submit-*-job* – Here * stands for the name of the LRMS. Currently supported LRMS are PBS/Torque, Condor, LoadLeveler, LSF, SLURM, and SGE. Also *fork* pseudo-LRMS is supported for testing purposes. This module is responsible for job submission to the LRMS.
- *cancel-*-job* – This script is for canceling jobs which have been already submitted to the LRMS.
- *scan-*-job* -This shell script is responsible for notifying the A-REX about completion of jobs. It's implementation for PBS uses server logs to extract information about jobs. If logs are not available it uses the less reliable *qstat* command for that. Other backends use different techniques.

In addition, there is also an administration utility:

- *gm-jobs* – prints a list of jobs available on the cluster and the number of jobs in each state.
  gm-jobs [-h] [-s] [-l] [-u uid] [-U name] [-c conf_file] [-d control_dir]
  -h – print short help,
  -s – print summary of jobs in each transfer share,
  -l – print more information about each job,
  -u – pretend utility is run by user with id *uid*,
  -U – pretend utility is run by user with name *name*,
  -c – use specified configuration file,
  -d – read information from specified control dir.

5.5 Common tasks

In this section the sysadmin will find some acknowledged ways of performing common tasks on an ARC CE. The information gathered here has been collected over time by ARC experts to fulfill the needs of the communities using ARC.

Tags on each task will show what is the related area of expertise.

5.5.1 How to ban a single user based on his/her DN?

Tags: Security, Authorization

This task can be performed in two ways, by using the **[vo]** and **[group]** blocks in arc.conf.

Solution 1 This solution does not require to restart A-REX.

1. Create a file containing the DNs of the users to ban, say, `/etc/grid-security/banned`, one per line.
2. in the **[group]** section, add a line:

   ```
   -file=/etc/grid-security/banned
   ```

   Remember that the rules are processed in order of comparison, so this rule must appear *before* a rule that will allow users to access the cluster. See **6.1.3**, *Commands in the [group] section* for a detailed explanation of the rule parsing process.

Solution 2 This solution requires restart of A-REX.

1. Create a file containing the DNs of the users to ban, say, `/etc/grid-security/banned`, one per line.
2. Choose a non existing unix user id, that is, a user not present in your system. Let's call it `unexistinguid`.
3. Create a special vo and group named "banned" in this way:

   ```
   [vo]
   id="banned_vo"
   vo="banned_vo"
   source="file:///etc/grid-security/banned"
   mapped_unix_id="unexistinguid"

   [group/banned_group]
   name="banned_group"
   vo="banned_vo"
   ```

   Be sure to have the `banned_group` *before* any other group in your arc.conf, or the user will never be rejected due to the group rule processing order. See **6.1.3**, *Commands in the [group] section* for a detailed explanation of the rule parsing process.

4. Restart A-REX service as in **5.1.2**, *Starting the CE*.

The user will be accepted but its jobs will not be processed as it will be assigned to a non-existing uid.

Chapter 6

Technical Reference

6.1 Reference of the `arc.conf` configuration commands

6.1.1 Generic commands in the [common] section

x509_user_key=*path* – sets the path to the host private key, usually `/etc/grid-security/hostkey.pem`

x509_user_cert=*path* – sets the path to the host public certificate, usually `/etc/grid-security/hostcert.pem`

x509_cert_dir=*path* – sets the path to the CA certificates, usually `/etc/grid-security/certificates`

gridmap=*path* – the path of the "grid map file", which maps Grid users to local unix accounts. This has to be set even if the mappings are dynamically created by the `nordugrid-arc-gridmap-utils` package is installed (see sections Section **4.4.1**, *Access control: users, groups, VOs* and Section **6.13**, *Structure of the grid-mapfile* for a brief explanation).

hostname=*hostname* – sets the hostname of the front-end. *hostname* is just a FQDN string. If not specified, *hostname* will be the one returned by the shell command `hostname -f`. Make sure this hostname is the same listed in `/etc/hosts` (see also Section **4.1.4**, *Networking*). This hostname has to be the same FQDN in the host certificates (see also Section **3.3**, *Installation of certificates*).

6.1.2 Commands in the [vo] section

These sections are also used by the `nordugridmap` utility which reads sources and generates list of Grid users belonging to particular VO or some other group.

vo=*vo_name* – specifies name of VO. It is required.

id=*unique_id* – defines an unique id for the VO.

file=*path* – path to file which contains list of users' DNs belonging to VO and their mappings. This file follows the format stated in Section **6.13**, *Structure of the grid-mapfile*. If `nordugridmap` is used it fills that file.

source=*URL* – specifies the URL from which a list of users may be obtained. There can be more than one `source` entries in the same `[vo]` section. URL is in the form $< protocolname : // < path >$ where $< protocolname >$ is one of: *vomss, http, https, ldap, file* and $< path >$ is a path in the form accepted by the protocol standard. In production environments, this URL to source files can be requested to the Grid organization who hosts the CA or the Grid computing organizations the CE is meant to be part of.

Some examples:

```
source="http://www.nordugrid.org/community.dn"
```

```
source="vomss://sample.hep.lu.se:8443/voms/knowarc.eu?/knowarc.eu"
source="file:///etc/grid-security/local-grid-mapfile"
```

mapped_unixid=*uid* – This is the local UNIX user account to which the DNs contained in the `source` command will be mapped. Only one `mapped_unixid` can be defined per `[vo]` section!

require_issuerdn=*[yes|no]* – *yes* would map only those DNs obtained from the urls which have the corresponding public CA packages installed. Default is *no*.

6.1.3 Commands in the [group] section

The [group] sections and subsections define authorization unities called groups.

name=*group_name* – specifies the name of an authorization group. If used within a `[group/subsection]` it has to be the same as the subsection name. If this command is omitted, `name` will implicitly taken from the subsection name.

Authorization is performed by applying a set of rules to users credentials. Credentials are certificates or certificates content (DN subject name, VO the user belongs to, CA that released the certificate...). Rules have the same $< command >=< value >$ format as rest of configuration file, with the difference that each rule command is prepended with optional modifiers: [+|-][!].

The rules are process sequentially in same order as presented in configuration. Processing stops at first matched rule.

A rule is said to *match* a credential if the credential **"satisfies"** the value specified by the command. By prepending rule with **!** matching is reversed. Matching rules turns into non-matching and non-matching into matching.

There are two kinds of matching. Rule prepended by + sign is called to produce *positive match* and matched credentials are considered to be belonging to this group. If rule is prepended with **-** sign it produces *negative match* and credentials are considered *not* belonging to this group. In both cases processing of rules for this groups is stopped. By default rule produces positive match - so + is optional.

Examples:

vo=*TESTVO* – This rule matches all the users belonging to the *TESTVO* Virtual Organization.

!vo=*TESTVO* – This rule matches all the users NOT belonging to the *TESTVO* Virtual Organization.

A credential (and therefore the user presenting it) can be *accepted* or *rejected*.

Accepted means that the credential becomes member of the group being processed - *positive match*.

Rejected means that the credential does **not** become member of group being processed - *negative match*.

Examples:

+vo=*TESTVO* – all the users belonging to the TESTVO Virtual Organization are Accepted into group. It can also be written as `vo = TESTVO`

-vo=*TESTVO* – all the users belonging to the TESTVO Virtual Organization are Rejected from this group.

+!vo=*TESTVO* – all the users NOT belonging to the TESTVO Virtual Organization are Accepted into group. It can also be written as `!vo = TESTVO`

-!vo=*TESTVO* – all the users NOT belonging to the TESTVO Virtual Organization are Rejected from group.

Note that `-vo = TESTVO` and `+!vo = TESTVO` do *not* do same thing. In first case user is rejected from group imeediately. In second case following rules will be processed - if any - and user may finally be accepted.

A summary of the modifiers is on **6.1**, *Basic Access Control modifiers and their meaning*.

Group membership does not automatically mean user is allowed to access resources served by A-REX. Whenever a GRID user submits a job to or requests information from the CE, A-REX will try to find a rule that matches that credential, for every [group...] section. Groups and rules will be processed in the order they appear in the `arc.conf` file.

Processing of rules in every group stops after the first positive or negative match, or when failure is reached. All groups are always procesed. Failures are rule-dependent and may be caused by conditions like missing files, unsupported or mistyped rule, etc.

The following *rule words* and arguments are supported:

subject=*subject [subject [...]]* – match user with one of specified subjects

file=*[filename [...]]* – read rules from specified files. Format of file similar to format of commands in *group* section with = replaces with space. Also in this file subject becomes default command and can be omited. So it becomes possible to use files consisting of only subject names of user credentials and Globus grid-mapfiles can be used directly.

remote=*[ldap://host:port/dn [...]]* – match user listed in one of specified LDAP directories (uses network connection hence can take time to process)

voms=*vo group role capabilities* – accept user with VOMS proxy with specified vo, group, role and capabilities. * can be used to accept any value.

vo=*[vo [...]]* – match user belonging to one of specified Virtual Organizations as defined in *vo* section configuration section (see `[vo]` above). Here VO membership is determined from corresponding *vo* section by comparing subject name of credentials to one strored in VO list file.

group=*[groupname [groupname [...]]]* – match user already belonging to one of specified groups.

plugin=*timeout plugin [arg1 [arg2 [...]]]* – run external plugin (executable or function in shared library) with specified arguments. Execution of plugin may **not** last longer than *timeout* seconds.
If plugin looks like `function@path` then function `int function(char*,char*,char*,...)` from shared library `path` is called (timeout has no effect in that case). Rule matches if plugin or executable exit code is 0. Following substitions are applied to arguments before plugin is started:

- %D - subject of userss cerificate,
- %P - name of credentials proxy file.

lcas=*library directory database* – - call LCAS functions to check rule. Here *library* is path to shared library of LCAS, either absolute or relative to *directory*; *directory* is path to LCAS installation directory, equivalent of LCAS_DIR variable; *database* is path to LCAS database, equivalent to LCAS_DB_FILE variable. Each arguments except library is optional and may be either skiped or replaced with *.

all accept any user

Here is an example of authorization group:

```
(1)    [group/admins]
(2)    -subject=/O=Grid/OU=Wrong Place/CN=Bad Person
(3)    file=/etc/grid-security/internal-staff
(4)    voms=nordugrid admin * *
```

The processing will work in the following way:

! invert matching. Match is treated as non-match. Non-match is treated as match, either positive (+ or nothing) or negative (-).

+ accept credential if matches following rule (positive match, default action);

- reject credential if matches following rule (negative match);

Figure 6.1: Basic Access Control modifiers and their meaning

Let credential has subject `/O=Grid/OU=Wrong Place/CN=Bad Person`. Then, subject matches (1) and the credential is Rejected from this group, processing of this group will stop.

Let credential has subject `/O=Grid/OU=Internal-Staff/CN=Good Person`, and let this subject be inside the file `/etc/grid-security/internal-staff`. Then, (1) doesn't match, processing continues to (2). Since subject is present inside the file specified by the `file` command, then the credential is Accepted in this group and the processing of this group stops.

Let credential has subject `/O=Grid/OU=SomeoneNotStaffButInnordugridVO/CN=Loyal Person`, and supplied credentials contain VOMS extension issued by `nordugrid` VO with group `admin` assigned. Let this credential be NOT present in the `internal-staff` file. Then, neither (1) nor (2) match and processing passes to (3). Since the credential belongs to that VO and group matches, the credential is Accepted in this group and processing of this group stops.

Let credential be `/O=Grid/OU=SomeOrg/CN=UN Known`, not present in the file neither belonging to VO. Processing passes through (1), (2), (3) without matching. Credential is Rejected from this group.

6.1.4 Commands in the [gridftpd] section

This section provides only short explanation of configuration options used in [gridftpd] section. For more detailed description see dedicated documentation [**?**].

daemon=*yes|no* – defines if GFS must run in daemon mode. Default is `yes`.

logfile=*path* – specifies log file for GFS. Default is not to have log file.

logsize=*size [number]* – specifies maximal `size` of log file in bytes. Optionally number of backup log files - files with same name as log file with .0, .1, etc. appended. If this command is specified GFS performs log rotation every time log size reaches specified vale. Default is not to perform log rotation.

logreopen=*yes|no* – defines if GFS closes log file after every write into log file and reopens it for every write. Default is `no`.

user=*username[:groupname]* – tell GFS to switch to specified user `username` and optionally to group `groupname` after start. Default is not to change user account.

pidfile=*path* – file containing the PID of the gridftpd process. Default is not to create such file.

debug=*number* – defines numerical - from 0 to 5 - verbosity of messages written into log file. Default is 3.

pluginpath=*path* – non-standard location of plugins location directory. Default is ARC_LOCATION/lib/arc.

port=*number* – specifies TCP/IP port number. Default is 2811.

maxconnections=*number* – limits number of sumultaneously served clients. Default is 100. Specifying 0 removes this limit. Connections over limit are rejected.

defaultbuffer=*number* – defines size of every buffer for data reading/writing. Default is 64kB. Actual value may decrease if cumulative size of all buffers exceedes value specified by `maxbuffer`.

maxbuffer=*number* – defines maximal amount of memory in bytes to be allocated for all data reading/writing buffers. Default is 640kB. Number of buffers is parallelism level requested by connecting client multiplied by 2 and increased by 1. Final number is limited by 41 from top and 3 from bottom. Hence even without parallel streams enable number of buffer will be 3.

firewall=*hostname* – defines hostname or IP address of firewall interface in case of GFS situated behind firewall with Network Address Translation functionality. If this command is specified GFS will use corresponding IP address instead of IP address of interface used for accepting client request in respose to PASV and similar commands.

encryption=*yes|no* – specifies if encryption of data channel is allowed. Default is `yes`. Data encryption is heavy operation.

allowunknown=*yes|no* – if set to `no` all clients with subject not in grid-mapfile are immediately rejected. If set to `yes`, this check is not performed.

unixmap/unixgroup/unixvo=*rule* – defines Grid identity to local account mapping rules. See [**?**] for more information.

voms_processing=*relaxed|standard|strict|noerrors* – specifies how to behave if failure happens during VOMS processing. See description of this option in 6.1.11.2 section.

Configuring the GridFTP plugins in a [gridftpd/subsection] subsection:

path=*path* – virtual path to which the service will be associated

plugin=*library_name* – use plugin library_name to serve virtual path.

groupcfg=*[group [group [...]]]* – defines authorization groups which are allowed to use functionality of this plugin. Default is to allow any client.

The GFS comes with 3 plugins: `fileplugin.so`, `gaclplugin.so` and `jobplugin.so`.

For the computing element, only the **jobplugin.so** is needed. It supports the following options:

configfile=*path* – defines non-standard location of the arc.conf file

allownew=*yes|no* – specifies if new jobs can be submitted. Default is yes

unixgroup/unixvo/unixmap=*rule* – - same options as in the top-level GFS configuration. If the mapping succeeds, the obtained local user will be used to run the submitted job.

remotegmdirs=*control_dir session_dir [drain]* – specifies control and session directories under the control of another A-REX to which jobs can be assigned (see [**?**], section 5) Remote directories can be added and removed without restarting the GFS. However, it may be desirable to drain them prior to removal by adding the drain option. In this case no new jobs will be assigned to these directories but their contents will still be accessible.

maxjobdesc=*size* – specifies maximal allowed size of job description in bytes. Default value is 5MB. If value is missing or set to 0 no limit is applied.

6.1.5 Commands in the [infosys] section

The `user` command here defines the UNIX user ID with which the slapd server, the infoproviders, BDII and registration scripts will run.

oldconfsuffix=*.suffix* – sets the suffix of the backup files of the low-level slapd config files in case the they are regenerated. Default is ".oldconfig".

overwrite_config=*yes|no* – determines if the grid-infosys startup script should generate new low-level slapd configuration files. By default the low-level configuration files are regenerated with every server startup making use of the values specified in the arc.conf.

hostname=*FQDN* – the hostname of the machine running the slapd service.

port=*port_number* – the port number where the slapd service runs. Default infosys port is 2135.

debug=*0|1* – sets the debug level/verbosity of the startup script. Default is 0.

slapd_loglevel=*verbosity_level* – sets the native slapd syslog loglevel (see man slapd for verbosity_level values). The default is set to no-logging (0) and it is RECOMMENDED not to be changed in a production environment. Non-zero slap_loglevel value causes serious performance decrease.

slapd_hostnamebind=**|* – may be used to set the hostname part of the network interface to which the slapd process will bind. Most of the cases no need to set since the hostname config parameter is already sufficient. The default is empty, but this can lead to problems in systems where slapd is set by security policies to be run only on the localhost interface. The wildcard will bind the slapd process to all the network interfaces available on the server. However, this is not recommended in clusters with many network interfaces. The recommended setting is the hostname assigned to the interface that will be publicly accessible.

threads=*num_threads* – the native slapd threads parameter, default is 32. If you run an Index Service too (see []) you should modify this value.

timelimit=*seconds* – the native slapd timelimit parameter. Maximum number of seconds the slapd server will spend answering a search request. Default is 3600.

providerlog=*path* – Specifies log file location for the information provider scripts. Default is `/var/log/infoprovider.log`.

provider_loglevel=*[0-5]* – loglevel for the infoprovider scripts (0, 1, 2, 3, 4, 5). The default is 1 (critical errors are logged). This corresponds to different verbosity levels, from less to maximum, namely: FATAL, ERROR, WARNING, INFO, VERBOSE, DEBUG

registrationlog=*path* – specifies the logfile for the registration processes initiated by your machine. Default is `/var/log/inforegistration.log`. For registration configuration, see Section **4.4.4**, *How to join the grid: registering to an index service.*

infosys_nordugrid=*enable|disable* – Activates or deactivates NorduGrid infosys schema [**?**] data generation and publishing. Default is *enabled*. This schema doesn't need further configuration.

infosys_glue12=*enable|disable* – Activates or deactivates Glue 1.x[] infosys schema data generation and publishing. Default is *disabled*. For configuration of this schema, see Section **4.5.1**, *Enabling or disabling LDAP schemas.*

infosys_glue2_ldap=*enable|disable* – Activates or deactivates Glue 2[**?**] infosys schema data generation and publishing. Default is *disabled*. For configuration of this schema, see Section **4.5.1**, *Enabling or disabling LDAP schemas.*

6.1.6 Commands in the [infosys/admindomain] section

Commands in this subsection are used to fill the AdminDomain GLUE2 data. This extends and generalizes the glue12 "site" concept.

name=*domain_name* – The string that identifies uniquely a domain name. Case is considered. This is mandatory if the [infosys/admindomain] block is enabled.

description=*text* – A human-readable description of the domain. Optional.

www=*domain_url* – A url pointing to relevant Domain information. Optional.

distributed=*yes|no* – A flag indicating the nature of the domain. Yes if services managed by the AdminDomain are considered geographycally distributed by the administrator themselves. Most likely this has to be set to no only if the CE is standalone and not part of other organizations. Optional

owner=*string* – A string representing some person or legal entity wich pays for the services or resources. Optional.

otherinfo=*text* – This field is only for further development purposes. It can fit all the information that doesn't fit above.

6.1.7 Commands in the [infosys/glue12] section

All the commands are mandatory if infosys_glue12 is enabled in the `[infosys]` section.

resource_location=*City, Country* – The field is free text but is a common agreement to have the City and the Country where the CE is located, separated by comma.

resource_latitude=*latitudevalue* – latitude of the geolocation where the CE is, expressed as degrees, e.g. 55.34380

resource_longitude=*longitudevalue* – latitude of the geolocation where the CE is, expressed as degrees, e.g. 12.41670

cpu_scaling_reference_si00=*number* – number represent the scaling reference number wrt si00. Please refer to the GLUE schema specification [] to know which value to put.

processor_other_description=*string* – String representing information on the processor, i.e. number of cores, becnhmarks.... Please refer to the GLUE schema specification [] to know which value to put. Example: `Cores=3,Benchmark=9.8-HEP-SPEC06`

glue_site_web=*url* – full url of the website of the site running the CE. Example: `http://www.ndgf.org`

glue_site_unique_id =*siteID* – Unique ID of the site where the CE runs. Example: `NDGF-T1`

provide_glue_site_info=*true|false* – This variable decides if the GlueSite should be published, in case a more complicated setup with several publishers of data to a GlueSite is needed.

6.1.8 Commands in the [cluster] section

These commands will affect the GLUE2 ComputingService, ComputingManager, ExecutionEnvironment, Policies objects, in how the information providers will fetch information about the frontend and computing nodes managed by the LRMS.

For a decent brokering, at least *architecture*, *nodecpu*, *nodememory* and *opsys* should be published.

cluster_alias=*name* – an arbitrary alias name of the cluster, optional

comment=*text* – a free text field for additional comments on the cluster in a single line, no newline character is allowed!

lrmsconfig=*description* – an optional free text field to describe the configuration of your Local Resource Management System (batch system).

homogeneity=*True|False* – determines whether the cluster consists of identical NODES with respect to cputype, memory, installed software (opsys). The frontend is NOT needed to be homogeneous with the nodes. In case of inhomogeneous nodes, try to arrange the nodes into homogeneous groups assigned to a queue and use queue-level attributes. Default is True. If set to False, the infosystem will try to fill GLUE2 ExecutionEnvironment information for each inhomogeneous node.

architecture=*string|adotf* – sets the hardware architecture of the NODES. The "architecture" is defined as the output of the "uname -m" (e.g. i686). Use this cluster attribute if only the NODES are homogeneous with respect to the architecture. Otherwise the queue-level attribute may be used for inhomogeneous nodes. If the frontend's architecture agrees to the nodes, the "adotf" (Automatically Determine On The Frontend) can be used to request automatic determination.

opsys=*string|adotf* – this multivalued attribute is meant to describe the operating system of the computing NODES. Set it to the opsys distribution of the NODES and not the frontend! opsys can also be used to describe the kernel or libc version in case those differ from the originally shipped ones. The distribution name should be given as distroname-version.number, where spaces are not allowed. Kernel version should come in the form kernelname-version.number. If the NODES are inhomogeneous with respect to this attribute do NOT set it on cluster level, arrange your nodes into homogeneous groups assigned to a queue and use queue-level attributes.

nodecpu=*string|adotf* – this is the cputype of the homogeneous nodes. The string is constructed from the /proc/cpuinfo as the value of "model name" and "@" and value of "cpu MHz". Do NOT set this attribute on cluster level if the NODES are inhomogeneous with respect to cputype, instead arrange the nodes into homogeneous groups assigned to a queue and use queue-level attributes. Setting the nodecpu="adotf" will result in Automatic Determination On The Frontend, which should only be used if the frontend has the same cputype as the homogeneous nodes. String can be like: "AMD Duron(tm) Processor @ 700 MHz"

nodememory=*number_MB* – this is the amount of memory (specified in MB) on the node which can be guaranteed to be available for the application. Please note in most cases it is less than the physical memory installed in the nodes. Do NOT set this attribute on cluster level if the NODES are inhomogeneous with respect to their memories, instead arrange the nodes into homogeneous groups assigned to a queue and use queue-level attributes.

defaultmemory=*number_MB* – If a user submits a job without specifying how much memory should be used, this value will be taken first. The order is: job specification → defaultmemory → nodememory → 1GB. This is the amount of memory (specified in MB) that a job will request(per rank).

nodeaccess= *|inbound|outbound* – determines how the nodes can connect to the internet. Not setting anything means the nodes are sitting on a private isolated network. "outbound" access means the nodes can connect to the outside world while "inbound" access means the nodes can

be connected from outside. inbound & outbound access together means the nodes are sitting on a fully open network.

cluster_location=*XX-postalcode* – The geographycal location of the cluster, preferably specified as a postal code with a two letter country prefix, like "DK-2100"

cluster_owner=*name* – it can be used to indicate the owner of a resource, multiple entries can be used

clustersupport=*email* – this is the support email address of the resource, multiple entries can be used

authorizedvo=*string* – a free-form string used to advertise which VOs are authorized on all the services on the CE. Multiple entries are allowed, each authorizedvo= entry will add a VO name to the infosystem. On the GLUE2 schema, this information will be published in the AccessPolicies and MappingPolicies. Example:

```
authorizedvo=VO:ATLAS
authorizedvo=support.nordugrid.org
```

6.1.9 Commands in the [queue] subsections

These commands will affect the ComputingShare GLUE2 object. Special GLUE2 MappingPolicies publishing configuration per queue is not yet supported.

fork_job_limit=*number*|cpunumber – sets the allowed number of concurrent jobs in a fork system, default is 1. The special value `cpunumber` can be used which will set the limit of running jobs to the number of cpus available in the machine. This parameter is used in the calculation of freecpus in a fork system.

name=*queuename* – The name of the grid-enabled queue, it must also be in the queue section name[queue/queuename]. Use "fork" for the fork LRMS.

homogeneity=*True*|*False* – - determines whether the queue consists of identical NODES with respect to cputype, memory, installed software (opsys). In case of inhomogeneous nodes, try to arrange the nodes into homogeneous groups and assigned them to a queue. Default is `True`.

scheduling_policy=*FIFO*|*MAUI* – this optional parameter tells the scheduling policy of the queue, PBS by default offers the FIFO scheduller, many sites run the MAUI. At the moment FIFO & MAUI is supported. If you have a MAUI scheduller you should specify the "MAUI" value since it modifies the way the queue resources are calculated. BY default the "FIFO" sceduller is assumed. More about this in chapter Section **4.4.2**, *Connecting to the LRMS*.

comment=*text* – a free text field for additional comments on the queue in a single line, no newline character is allowed!

6.1.10 Commands in the [infosys/cluster/registration/registrationname] subsections

Computing resource (cluster) registration block, configures and enables the registration process of a Computing Element to an Index Service. The string `infosys/cluster/registration/` identifies the block, while `registrationname` is a free form string used to identify a registration to a specific index. A cluster can register to several Index Services. In this case, each registration process should have its own block, each with its own `registrationname`.

Registration commands explained:

targethostname=*FQDN* – The FQDN of the host running the target index service.

targetport=*portnumber* – Port where the target Index Service is listening. Defaults to 2135.

targetsuffix=*ldapsuffix* – ldap suffix of the target index service. This has to be provided by a manager of the index service, as it is a custom configuration value of the Index Service. Usually is a string of the form "mds-vo-name=<custom value>,o=grid"

regperiod=*seconds* – the registration script will be run each number of *seconds*. Defaults to 120.

registranthostname=*FQDN* – the registrant FQDN. This is optional as ARC will try to guess it from the system of from then [common] block. Example: `registranthostname="myhost.org"`

registrantport=*port* – the port where the local infosystem of the registrant is running. Optional, as this port is already specified in the [infosys] block. Example: `registrantport="2135"`

registrantsuffix=*ldap_base_string* – the LDAP suffix of the registrant cluster resource. Optional, as it is automatically determined from the [infosys] block and the registration blockname. In this case the default registrantsuffix will be:

"nordugrid-cluster-name=FQDN,Mds-Vo-name=local,o=Grid"

Please mind uppercase/lowercase characters above if defining allowreg in an index! Don't set it unless you want to overwrite the default. Example:
`registrantsuffix="nordugrid-cluster-name=myhost.org,Mds-Vo-name=local,o=grid"`

6.1.11 Commands in the [grid-manager] section

6.1.11.1 Commands affecting the A-REX process and logging

pidfile=*path* – specifies file where process id of A-REX process will be stored.
Defaults to `/var/run/arched-arex.pid` if running as root and `$HOME/arched.pid` otherwise.

logfile=*path* – specifies name of file for logging debug/informational output.
Defaults to `/var/log/arc/grid-manager.log`. Note: if installed from binary packages, ARC comes with configuration for *logrotate* log management utility and A-REX log is managed by *logrotate* by default.

logsize=*size number* – restricts log file size to *size* and keeps *number* archived log files. This command enables log rotation by ARC and should only be used if *logrotate* or other external log rotation utility is not used. Using ARC log rotation and external log management simultaneously may result in strange behaviour.

logreopen=*yes|no* – specifies if log file must be opened before writing each record and closed after that. By default log file is kept open all the time (default is no).

debug=*number* – specifies level of debug information. More information is printed for higher levels. Currently the highest effective number is 5 (DEBUG) and lowest 0 (FATAL). Defaults to 2 (WARNING).

user=*username[:groupname]* – specifies username and optionally groupname to which the A-REX must switch after reading configuration. Defaults to *not switch*.

6.1.11.2 Commands affecting the A-REX Web Service communication interface

voms_processing=*relaxed|standard|strict|noerrors* – specifies how to behave if failure happens during VOMS processing.

- relaxed – use everything that passed validation.
- standard – same as relaxed but fail if parsing errors took place and VOMS extension is marked as critical. This is a default.
- strict – fail if any parsing error was discovered.
- noerrors – fail if any parsing or validation error happened.

Default is *standard*. This option is effective only if A-REX is started using startup script.

arex_mount_point=*URL* – specifies URL for accessing A-REX through WS interface. This option is effective only if A-REX is started using startup script. The presence of this option enables WS interface. Default is to not provide WS interface for communication with A-REX.

enable_emies_interface=*yes|no* – turns on or off the EMI Execution Service interface. If enabled the interface can be accessed at the URL specified by *arex_mount_point* (this option must also be specified). Default is *off*.

max_job_control_requests=*number* – specifies maximal number of simultaneously processed job control requests. Requests above that threshold are put on hold and client is made to wait for response. Default value is 100. Setting value to -1 turns this limit off. This option is effective only if A-REX is started using startup script.

max_infosys_requests=*number* – specifies maximal number of simultaneously processed information requests. Requests above that threshold are put on hold and client is made to wait for response. Default value is 1. Setting value to -1 turns this limit off. This option is effective only if A-REX is started using startup script.

max_data_transfer_requests=*number* – specifies maximal number of simultaneously processed data read/write requests. Requests above that threshold are put on hold and client is made to wait for response. Default value is 100. Setting value to -1 turns this limit off. This option is effective only if A-REX is started using startup script.

6.1.11.3 Commands setting limits

maxjobs=*max_processed_jobs [max_running_jobs [max_jobs_per_dn [max_jobs_total]]]* – specifies maximum number of jobs being processed by the A-REX at different stages:
max_processed_jobs – maximum number of concurrent jobs processed by A-REX. This does not limit the amount of jobs which can be submitted to the cluster.
max_running_jobs – maximum number of jobs passed to Local Resource Management System
max_jobs_per_dn – maximum number of concurrent jobs processed by A-REX per user DN. If this option is used the total maximum number of jobs processed is still *max_processed_jobs*.
max_jobs_total – total maximum number of jobs associated with service. It is advised to use this limit only inexceptional case because it also accounts for finished jobs.

Missing value or -1 means no limit.

maxload=*max_frontend_jobs [emergency_frontend_jobs [max_transferred_files]]* – specifies maximum load caused by data staging being processed on frontend:
max_frontend_jobs – maximum number of jobs in PREPARING and FINISHING states (downloading and uploading files). Jobs in these states can cause a heavy load on the A-REX host. This limit is applied before moving jobs to PREPARING and FINISHING states.
emergency_frontend_jobs – if limit of *max_frontend_jobs* is used only by PREPARING or by FINISHING jobs, aforementioned number of jobs can be moved to another state. This is used to avoid the case where jobs cannot finish due to a large number of recently submitted jobs.
max_transferred_files – maximum number of files being transferred in parallel by every job. Used to decrease load on not so powerful frontends.
Missing value or -1 means no limit.

maxloadshare=*max_share share_type* – specifies a sharing mechanism for data transfer. *max_share* is the maximum number of processes that can run per transfer share and *share_type* is the scheme used to assign jobs to transfer shares. See Section **6.6**, *Transfer shares* for possible values and more details.

share_limit=*name limit* – specifies a transfer share that has a number of processes different from the default value in maxloadshare. *name* is the name of the share and *limit* is the number of processes for this share. In the configuration it should appear after maxloadshare. Can be repeated several times for different shares. See Section **6.6**, *Transfer shares* for how to compose shares' names and more details.

6.1.11.4 Commands related to file staging

securetransfer=*yes|no* – specifies whether to use encryption while transferring data. Currently works for GridFTP only. Default is *no*. It may overridden for every source/destination by values specified in URL options.

passivetransfer=*yes|no* – specifies whether GridFTP transfers are passive. Setting this option to yes can solve transfer problems caused by firewalls. Default is no.

localtransfer=*yes|no* – specifies whether to pass file downloading/uploading task to computing node. If set to yes the A-REX will not download/upload files but compose script submitted to the LRMS in order that the LRMS can execute file transfer. This requires installation of A-REX and all related software to be accessible from computing nodes and environment variable ARC_LOCATION to be set accordingly. Default is *no*.

speedcontrol=*min_speed min_time min_average_speed max_inactivity* – specifies how long or slow data transfer is allowed to take place. Transfer is canceled if transfer rate (bytes per second) is lower than *min_speed* for at least *min_time* seconds, or if average rate is lower than *min_average_speed*, or no data is received for longer than *max_inactivity* seconds. To allow statistics to build up, no transfers will be stopped within the first 3 minutes.

preferredpattern=*pattern* – specifies how to order multiple replicas of an input file according to preference. It consists of one or more patterns (strings) separated by a pipe character (|) listed in order of preference. Input file replicas will be matched against each pattern and then ordered by the earliest match. If the dollar character ($) is used at the end of a pattern, the pattern will be matched to the end of the hostname of the replica.

newdatastaging=*yes|no* – turns on or off the new data staging framework* (also known as DTR), which replaces the downloader and uploader utilities. Default is *no*.

copyurl=*template replacement* – specifies that URLs starting from *template* should be accessed at *replacement* instead. The *template* part of the URL will be replaced with *replacement.* This option is useful when for example a Grid storage system is accessible as a local file system on the A-REX host. *replacement* can be either a URL or a local path starting from '/'. It is advisable to end template with '/'.

linkurl=*template replacement [node_path]* – mostly identical to *copyurl* but file will not be copied. Instead a soft-link will be created. *replacement* specifies the way to access the file from the frontend, and is used to check permissions. The *node_path* specifies how the file can be accessed from computing nodes, and will be used for soft-link creation. If *node_path* is missing, *local_path* will be used instead. Neither *node_path* nor *replacement* should be URLs.

NOTE: URLs which fit into *copyurl* or *linkurl* are treated as more easily accessible than other URLs. That means if A-REX has to choose between several URLs from which should it download input file, these will be tried first.

6.1.11.5 Commands related to usage reporting

jobreport=*URL ... number* – specifies that A-REX has to report information about jobs being processed (started, finished) to a remote service running at the given *URL*. Multiple entries and multiple URLs are allowed. *number* specifies how long (in days) old records have to be kept if failed to be reported. The last specified value becomes effective.

jobreport_publisher=*filename* – specifies the name of the executable which will be run by the A-REX periodically to publish job reports if a jobreport URL is specified. The executable will be searched in the nordugrid `libexec` directory. The default name is `jura`.

jobreport_credentials=*key_file [cert_file [ca_dir]]* – specifies the credentials for accessing the accounting service.

jobreport_options=*options* – specifies additional options for the usage reporter (e.g. JURA). See **4.4.5**, *Accounting with JURA*.

*see http://wiki.nordugrid.org/index.php/Data_Staging

6.1.11.6 Other general commands in the [grid-manager] section

wakeupperiod=*time* – specifies how often the A-REX checks for job state changes (like new arrived job, job finished in LRMS, etc.). *time* is a minimal time period specified in seconds. Default is *3 minutes*. The A-REX may also be woken up by external processes such as LRMS scripts before this time period expires.

authplugin=*state options plugin* – specifies *plugin* (external executable) to be run every time job is about to switch to *state*. The following states are allowed: ACCEPTED, PREPARING, SUBMIT, FINISHING, FINISHED and DELETED. If exit code is not 0 job is canceled by default. *Options* consists of *name=value* pairs separated by commas. The following *name*s are supported: *timeout* – specifies how long in seconds execution of the plugin allowed to last (mandatory, "*timeout=*" can be skipped for backward compatibility).
onsuccess, *onfailure* and *ontimeout* – defines action taken in each case (*onsuccess* happens if exit code is 0). Possible actions are:
pass – continue execution,
log – write information about result into log file and continue execution,
fail – write information about result into log file and cancel job. Default actions are *fail* for *onfailure* and *ontimeout*, *pass* for *onsuccess*.

localcred=*timeout plugin* – specifies *plugin* (external executable or function in shared library) to be run every time job has to do something on behalf of local user. Execution of *plugin* may not last longer than *timeout* seconds. If *plugin* looks like *function@path* then function *int function(char*,char*,char*,...)* from shared library *path* is called (*timeout* is not functional in that case). If exit code is not 0 current operation will fail. This functionality was introduced for acquiring Kerberos tickets for local filesystem access and is currently deprecated.

norootpower=*yes|no* – if set to *yes* all processes involved in job management will use local identity of a user to which Grid identity is mapped in order to access file system at path specified in ***session*** command (see below). Sometimes this may involve running temporary external process under specified account.

joblog=*path* – specifies where to store log file containing information about started and finished jobs. This file contains one line per every started and evety finished job. Default is not to write such file.

6.1.11.7 Per UNIX user commands and setting the control directory

The A-REX can serve multiple UNIX users with separate control directories, or it can serve all of them with the same control directory. The per UNIX user commands can be different for each UNIX user served. For each control directory the per UNIX user commands should come first, then finally the **control** command which starts serving a UNIX user based on the commands preceding it.

mail=*e-mail_address* – specifies an email address **from** which notification mails are sent.

defaultttl=*ttl [ttr]* – specifies the time in seconds for the SD to be available after job finishes (*ttl*). Second optional number (*ttr*) defines time since removal of the SD till all information about job is discarded - job stays in DELETED state during that period. Time is specified in seconds. Defaults are 7 days for *ttl* and 30 days for *ttr*. The minumum value for both parameters is 2 hours.

lrms=*lrms_name [default_queue_name]* – specifies names for the LRMS and queue.
Queue name can also be specified in the JD.

sessiondir=*path [drain]* – specifies the path to the directory in which the SD is created. Multiple session directories may be specified by specifying multiple *sessiondir* commands. In this case jobs are spread evenly over the session directories. If the path is * the default sessiondir is used - *$HOME/.jobs*. The * can be used only if corresponding *control* command refers to user know during service startup. When adding a new session directory, ensure to restart the A-REX so that jobs assigned there are processed. A session directory can be drained prior to removal by adding the *"drain"* option (no restart is required in this case if *gridftp* interface is used). No new jobs will be assigned to this session directory but running jobs will still be accessible. When all jobs are processed and the session directory is empty, it can be removed from configuration and the A-REX should be restarted.

cachedir=*path [link_path]* – specifies a directory to store cached data (see Section Section **6.5**, *Cache*). Multiple cache directories may be specified by specifying multiple *cachedir* commands. Cached data will be distributed over multiple caches according to free space in each. Specifying no *cachedir* command or commands with an empty path disables caching. The optional *link_path* specifies the path at which *path* is accessible on computing nodes, if it is different from the path on the A-REX host. If *link_path* is set to '.' files are not soft-linked, nor are per-job links created, but files are copied to the session directory. If a cache directory needs to be drained, then *cachedir* should specify *"drain"* as the *link_path*.

remotecachedir=*path [link_path]* – specifies caches which are under the control of other A-REXs, but which this A-REX can have read-only access to (see Section 6.5.3). Multiple remote cache directories may be specified by specifying multiple *remotecachedir* commands. If a file is not available in paths specified by cachedir, the A-REX looks in remote caches. *link_path* has the same meaning as in *cachedir*, but the special path "replicate" means files will be replicated from remote caches to local caches when they are requested.

cachesize=*high_mark [low_mark]* – specifies high and low watermarks for space used on the file system on which the cache directory is located, as a percentage of total file system capacity. When the max is exceeded, files will be deleted to bring the used space down to the min level. It is a good idea to have each cache on its own separate file system. If no *cachesize* is specified, or it is specified without parameters, no cleaning is done. These cache settings apply to all caches specified by *cachedir* commands.

cachelifetime=*lifetime* – if cache cleaning is enabled, files accessed less recently than the *lifetime* time period will be deleted. Example values of this option are 1800, 90s, 24h, 30d. When no suffix is given the unit is seconds.

cachelogfile=*path* – specifies the filename where output of the *cache-clean* tool should be logged. Defaults to */var/log/arc/cache-clean.log*.

cacheloglevel=*number* – specifies the level of logging by the *cache-clean* tool, between 0 (FATAL) and 5 (DEBUG). Defaults to 3 (INFO).

cachecleantimeout=*timeout* – the timeout in seconds for running the cache-clean tool. If using a large cache or slow file system this value can be increased to allow the cleaning to complete. Defaults to 3600 (1 hour).

maxrerun=*number* – specifies maximal number of times job will be allowed to rerun after it failed at any stage. Default value is *5*. This only specifies a upper limit. The actual number is provided in job description and defaults to 0.

maxtransfertries=*number* – specifies the maximum number of times download and upload will be attempted per file (retries are only performed if an error is judged to be temporary, for example a communication error with a remote service). This number must be greater than 0 and defaults to 10.

All per-user commands should be put in the configuration file before the *control* command which initiates serviced user.

control=*path username [username [...]]* – This option initiates local UNIX user as being serviced by the A-REX. The *path* refers to the control directory. If the path is * the default one is used – `$HOME/.jobstatus`. The *username* stands for UNIX name of the local user. Multiple names can be specified. If the name starts from @ rest is treated as path to file containing list of serviced users. Usernames are specified one per line and may be optionally prepended with Grid identity of user - last one is ignored. That is done for compatibility with so-called grid-mapfile (for more information please see the description of Globus project [**?**]). Also the special name '.'(dot) can be used as username. Corresponding control directory will be used for **any** user. In that case such *control* option should be the last *control* option in the configuration file.

controldir=*path* – This is identical to "***control***=`path .`" It presumes special username '.' and is always executed last independent of its placement in file. This option is provided for convenience for usual setups which have dynamic set of local serviced users and use same control directory for all them.

helper=*username command [argument [argument [...]]]* – associates an external pro-

gram with the local UNIX user. This program will be kept running under account of the user specified by *username*. Special names can be used: '*' – all names from /etc/grid-security/grid-mapfile, '.' – root user. The user should be already configured with *control* option (except root, who is always configured). *command* is an executable and *argument*s are passed as arguments to it. This executable is started under specified local account and re-started every time it exits.

6.1.11.8 Global commands specific to communication with the underlying LRMS

gnu_time=*path* – path to *time* utility.

tmpdir=*path* – path to directory for temporary files.

runtimedir=*path* – path to directory which contains *runtimenvironment* scripts.

shared_filesystem=*yes|no* – if computing nodes have an access to session directory through a shared file system like NFS.
Corresponds to an environment variable RUNTIME_NODE_SEES_FRONTEND.
(See Section **6.15**, *Environment variables set for the job submission scripts*).

nodename=*command* – command to obtain hostname of computing node.

scratchdir=*path* – path on computing node where to move session directory before execution. Default is not to move session directory before execution.

shared_scratch=*path* – path on frontend where ***scratchdir*** can be found. Only needed if *scratchdir* is used.

6.1.11.9 Substitutions in the command arguments

In the command arguments (paths, executables, ...) following substitutions can be used:

%R – session root – see command *sessiondir*

%C – control dir – see command *control*

%U – username (as specified in configuration, hence empty for '.' control directories)

%u – userid – numerical

%g – groupid – numerical

%H – home dir – home of username as specified in /etc/passwd

%Q – default queue – see command *lrms*

%L – lrms name – see command *lrms*

%W – installation path – ${ARC_LOCATION}

%F – path to configuration file of this instance

%I – job ID (for plugins only, substituted in runtime)

%S – job state (for *authplugin* plugins only, substituted in runtime)

%O – reason (for *localcred* plugins only, substituted in runtime). Possible reasons are:

new – new job, new credentials

renew – old job, new credentials

write – write/delete file, create/delete directory

read – read file, directory, etc.

extern – call external program

6.1.12 Commands in the [data-staging] section

The design of the new data staging framework is detailed in Section **6.3**, *The new data staging framework*.

maxdelivery=*number* – maximum number of files in delivery, i.e. the maximum number of physical transfer slots.

maxprocessor=*number* – maximum number of files in each processing state. These states are for example checking the cache or resolving replicas in index services.

maxemergency=*number* – maximum emergency slots for delivery and processor. If the maximum slots as defined above are already used, a new transfer share can still proceed by using an emergency slot. These slots therefore stop shares from being blocked by others in busy situations.

maxprepared=*number* – maximum number of files prepared for transfer. For protocols such as SRM files are pinned for transfer. This parameter determines the maximum number of files that are pinned. It should normally be set a few times larger than maxdelivery so that new pinned transfers are ready to start when others finish, but not so high as to pass the limits of the storage systems. This number is applied per-transfer share and only for applicable protocols.

sharetype=*type* – sharing mechanism for data transfer. This is the scheme used to assign transfers to shares. See Section **6.6**, *Transfer shares* for possible values and more details.

definedshare=*share priority* – a transfer share with a specific priority, different from the default. This is used to give more or less transfer slots to certain shares. `priority` should be a number between 1 and 100 (higher number is higher priority). The default priority for a share is 50.

deliveryservice=*url* – remote delivery service for executing transfers[†]

localdelivery=*yes|no* – in case remote delivery services are configured using the previous option, this option specifices whether or not delivery should also be done locally on the A-REX host. Default is *no*.

remotesizelimit=*number* – file size in bytes under which data transfer will always use local delivery rather than a remote delivery service. This can optimise performance when many small files are being transferred, where communication overhead with remote services can become a bottleneck.

usehostcert=*yes|no* – specifies whether to use the A-REX host certificate when contacting remote delivery services. This can be done for added security but only with host certificates which can be used as client certificates. Default is *no*.

dtrlog=*path* – file in which to periodically dump data staging information. This data can be used to monitor the system. Defaults to *controldir/dtrstate.log*.

6.1.13 PBS specific commands

For each grid-enabled (or Grid visible) PBS queue a corresponding `[queue/queuename]` subsection must be defined. `queuename` should be the PBS queue name.

lrms=*"pbs"* – enables the PBS batch system back-end

pbs_bin_path=*path* – in the `[common]` section should be set to the path to the qstat,pbsnodes,qmgr etc. PBS binaries.

pbs_log_path=*path* – in the `[common]` sections should be set to the path of the PBS server logfiles which are used by A-REX to determine whether a PBS job is completed. If not specified, A-REX will use the `qstat` command to find completed jobs.

lrmsconfig=*text* – in the `[cluster]` block can be used as an optional free text field to describe further details about the PBS configuration (e.g. `lrmsconfig="single job per processor"`). This information is then exposed through information interfaces.

dedicated_node_string=*text* – in the `[cluster]` block specifies the string which is used in the PBS node config to distinguish the Grid nodes from the rest. Suppose only a subset of nodes are

[†] `http://wiki.nordugrid.org/index.php/Data_Staging/Multi-host`

available for Grid jobs, and these nodes have a common `node property` string, this case the `dedicated_node_string` should be set to this value and only the nodes with the corresponding PBS `node property` are counted as Grid enabled nodes. Setting the `dedicated_node_string` to the value of the PBS `node property` of the grid-enabled nodes will influence how the totalcpus, user freecpus is calculated. No need to set this attribute if the cluster is fully available for the Grid and the PBS configuration does not use the `node property` method to assign certain nodes to Grid queues.

scheduling_policy=*FIFO|MAUI* – in the `[queue/queuename]` subsection describes the scheduling policy of the queue. PBS by default offers the FIFO scheduler, many sites run the MAUI. At the moment `FIFO` & `MAUI` are supported values. If you have a MAUI scheduler you should specify the `"MAUI"` value since it modifies the way the queue resources are calculated. By default the `"FIFO"` scheduler type is assumed.

maui_bin_path=*path* – in the `[queue/queuename]` subsection sets the path of the MAUI commands like `showbf` when `"MAUI"` is specified as `scheduling_policy` value. This parameter can be set in the `[common]` block as well.

queue_node_string=*text* – in the `[queue/queuename]` block can be used similar to the configuration command `dedicated_node_string`. In PBS you can assign nodes to a queue (or a queue to nodes) by using the `node property` PBS node configuration method and assigning the marked nodes to the queue (setting the `resources_default.neednodes = queue_node_string` for that queue). This parameter should contain the `node property` string of the queue-assigned nodes. Setting the `queue_node_string` changes how the queue-totalcpus, user freecpus are determined for this queue.

6.1.14 Condor specific commands

lrms=*"condor"* – in the `[common]` section enables the Condor batch system back-end.

condor_location=*path* – in the `[common]` section should be set to the Condor install prefix (i.e., the directory containing Condor's bin, sbin, etc).

condor_config=*text* – in the `[common]` section should be set to the value the environment variable `CONDOR_CONFIG` should have (but don't try to use the environment variable directly as `$CONDOR_CONFIG`, since it will probably not be defined when arc.conf is parsed!)

condor_rank=*ClassAd float expression* – in the `[common]` section, if defined, will cause the `Rank` attribute to be set in each job description submitted to Condor. Use this option if you are not happy with the way Condor picks out nodes when running jobs and want to define your own ranking algorithm. `condor_rank` should be set to a ClassAd float expression that you could use in the `Rank` attribute in a Condor job description. For example:

```
condor_rank="(1-LoadAvg/2)*(1-LoadAvg/2)*Memory/1000*KFlops/1000000"
```

condor_requirements=*constraint string* – in the `[queue/queuename]` section defines a subpool of condor nodes. Condor does not support queues in the classical sense. It is possible, however, to divide the Condor pool in several sub-pools. An ARC "queue" is then nothing more than a subset of nodes from the Condor pool.

Which nodes go into which queue is defined using the `condor_requirements` configuration option in the corresponding `[queue/queuename]` section. It's value must be a well-formed constraint string that is accepted by a `condor_status -constraint '...'` command. Internally, this constraint string is used to determine the list of nodes belonging to a queue. This string can get quite long, so, for readability reasons it is allowed to split it up into pieces by using multiple `condor_requirements` options. The full constrains string will be reconstructed by concatenating all pieces.

Queues should be defined in such a way that their nodes all match the information available in ARC about the queue. A good start is for the `condor_requirements` attribute to contain restrictions on the following: `Opsys`, `Arch`, `Memory` and `Disk`. If you wish to configure more than one queue, it's good to have queues defined in such a way that they do not overlap. In the following example disjoint memory ranges are used to ensure this:

```
[queue/large]
condor_requirements="(Opsys == "linux" && (Arch == "intel" || Arch == "x86_64")"
condor_requirements=" && (Disk > 30000000 && Memory > 2000)"
[queue/small]
condor_requirements="(Opsys == "linux" && (Arch == "intel" || Arch == "x86_64")"
condor_requirements=" && (Disk > 30000000 && Memory <= 2000 && Memory > 1000)"
```

Note that nodememory attribute in arc.conf means the maximum memory available for jobs, while the Memory attribute in Condor is the physical memory of the machine. To avoid swapping (and these are probably not dedicated machines!), make sure that nodememory is smaller than the minimum physical memory of the machines in that queue. If for example the smallest node in a queue has 1Gb memory, then it would be sensible to use nodememory="850" for the maximum job size.

In case you want more precise control over which nodes are available for Grid jobs, using pre-defined ClassAds attributes (like in the example above) might not be sufficient. Fortunately, it's possible to mark nodes by using some custom attribute, say NORDUGRID_RESOURCE. This is accomplished by adding a parameter to the node's local Condor configuration file, and then adding that parameter to STARTD_EXPRS:

```
NORDUGRID_RESOURCE = True
STARTD_EXPRS = NORDUGRID_RESOURCE, $(STARTD_EXPRS)
```

Now queues can be restricted to contain only "good" nodes. Just add to each [queue/queuename] section in arc.conf:

```
condor_requirements=" && NORDUGRID_RESOURCE"
```

6.1.15 LoadLeveler specific commands

lrms="*ll*" – in the [common] section enables the LoadLeveler batch system.

ll_bin_path=*path* – in the [common] section must be set to the path of the LoadLeveler binaries.

6.1.16 Fork specific commands

lrms="*fork*" – in the [common] section enables the Fork back-end. The queue must be named "fork" in the [queue/fork] subsection.

fork_job_limit=*cpunumber* – sets the number of running Grid jobs on the fork machine, allowing a multi-core machine to use some or all of its cores for Grid jobs. The default value is 1.

6.1.17 LSF specific commands

lrms="*lsf*" – in the [common] section enables the LSF back-end

lsf_bin_path=*path* – in the [common] section must be set to the path of the LSF binaries

lsf_profile_path=*path* – must be set to the filename of the LSF profile that the back-end should use.

Furthermore it is very important to specify the correct architecture for a given queue in arc.conf. Because the architecture flag is rarely set in the xRSL file the LSF back-end will automatically set the architecture to match the chosen queue. LSF's standard behaviour is to assume the same architecture as the frontend. This will fail for instance if the frontend is a 32 bit machine and all the cluster resources are 64 bit. If this is not done the result will be jobs being rejected by LSF because LSF believes there are no useful resources available.

6.1.18 SGE specific commands

lrms="*sge*" – in the [common] section enables the SGE batch system back-end.

sge_root=*path* – in the [common] section must be set to SGE's install root.

sge_bin_path=*path* – in the [common] section must be set to the path of the SGE binaries.

sge_cell=*cellname* – in the [common] section can be set to the name of the SGE cell if it's not the default

sge_qmaster_port=*port* – in the [common] section can be set to the qmaster port if the sge command line clients require the SGE_QMASTER_PORT environment variable to be set

sge_execd_port=*port* – in the [common] section can be set to the execd port if the sge command line clients require the SGE_EXECD_PORT environment variable to be set

sge_jobopts=*options* – in the [queue/queuename] section can be used to add custom SGE options to job scripts submitted to SGE. Consult SGE documentation for possible options.

6.1.19 SLURM specific commands

lrms="*sge*" – in the [common] section enables the SLURM batch system back-end.

slurm_bin_path=*path* – in the [common] section must be set to the path of the SLURM binaries.

6.1.20 Commands for the **urlogger** accounting component

Note that prefered way for handling accounting in A-REX is to use Jura components described in section 6.9.

These commands are in the [logger] section:

log_dir=*path* – sets the top directory for the generated usage records. The option will default to /var/spool/nordugrid/usagerecords/ and can be left out. We suggest you leave out this option, unless you have a reason not to.

log_all=*URL [URL...]* – configures where the usage records are sent to. All usage records will be registered to the URLs specified with the log_all option. It is possible to specifiy multiple URLs separated by a space. There can be only one log_all command.

log_vo=*VO URL [, VO URL...]* – makes it possible to only register usage records run with certain VO users to a given URL. It is possible to have multiple entries, by seperating entries with comma. There can be only one log_vo command.

ur_lifetime=*days* – specifies how many days usage records are kept after being archived. The default is 30.

urlogger_logfile=*path* – specifies the logfile the urlogger should write its logs. Default is /var/log/arc-ur-logger.log.

urlogger_loglevel=*debug|info|warning* – specifies the verbosity of logging.

registrant_logfile=*path* – specifies the logfile for the urlogger registration module, default is /var/log/arc-ur-registration.log.

6.2 Handling of the input and output files

One of the most important tasks of the A-REX is to take care of processing of the input and output data (files) of the job. Input files are gathered in the session directory (SD) or in the associated cache area. If caching is enabled, then the A-REX checks the cache whether a requested input file is already present (with proper authorization checks and timely invalidation of cached data), and links (or copies) it to the SD of the job without re-downloading it. If file is present in the cache but not marked as authorized for specific grid identity then connection to data server is performed for authorization check.

There are two ways to put a file into the SD:

- If a file is specified in the job description as an input file with a remote source location: the A-REX contacts the remote location (using the user's delegated credentials) and downloads the file into the session directory or cache location using one of the supported protocols (GridFTP, FTP, HTTP, HTTPS, and also communicating with Grid file catalogs is supported).
- If a file is specified in the job description as an input file without a remote source location: the A-REX expects the client tool to upload the file to the session directory (SD) using the URL provided by A-REX. The client tools should do this step automatically.

The A-REX does not provide any other reliable way to obtain input data.

After the job finishes, the files in the session directory are treated in three possible ways:

- If a file is specified in the job description as an output file with a remote target location: the A-REX uploads the results to the remote storage (optionally register the file to a catalog), then it will remove the file from the session directory. If the job execution fails, these files will not be uploaded (but they will be kept for the user to download). Depending of used submission interface and corresponding job description format it is possible to request more sophisticated conditions for files processing in case of failure.
- If a file is specified in the job description as an output file without a target location: the A-REX will keep the files, and the user can download them by accessing the session directory. The client tools usually support downloading these files.
- If a file is not specified in the job description as an output file: the A-REX will remove the file from the session directory after the job finished.

It is possible to specify in job description an option to keep a whole directory, but if a file is not specified in the job description as an output file and it is not in a directory which is requested to be kept, it will be removed when the job is finished.

6.3 The new data staging framework

After a growing number of issues with the current ARC data management model, it was decided that an entirely new framework should be designed. The new data staging framework uses a three-layer architecture, shown in Figure 6.2.

The Generator uses user input of tasks to construct a Data Transfer Request (DTR) per file that needs to be transferred. These DTRs are sent to the Scheduler for processing. The Scheduler sends DTRs to the Pre-processor for anything that needs to be done up until the physical transfer takes place (e.g. cache check, resolve replicas) and then to Delivery for the transfer itself. Once the transfer has finished the Post-processor handles any post-transfer operations (e.g. register replicas, release requests). The number of slots available for each component is limited, so the Scheduler controls queues and decides when to allocate slots to specific DTRs, based on the prioritisation algorithm implemented. This layered architecture allows any implementation of a particular component to be easily substituted for another, for example a GUI with which users can enter DTRs (Generator) or an external point to point file transfer service (Delivery).

To enable the new framework the following option should be present in the `[grid-manager]` section of arc.conf:

```
newdatastaging=yes
```

In A-REX version 2.0 a dedicated configuration section `[data-staging]` was introduced - see Section 6.1.12. However, if this section is not used then some configuration can be taken from existing options in `[grid-manager]`. The maximum number of delivery, pre-processor and post-processor slots is taken from the "maxload" option (max frontend jobs * max transferred files). Setting options in the `[data-staging]`

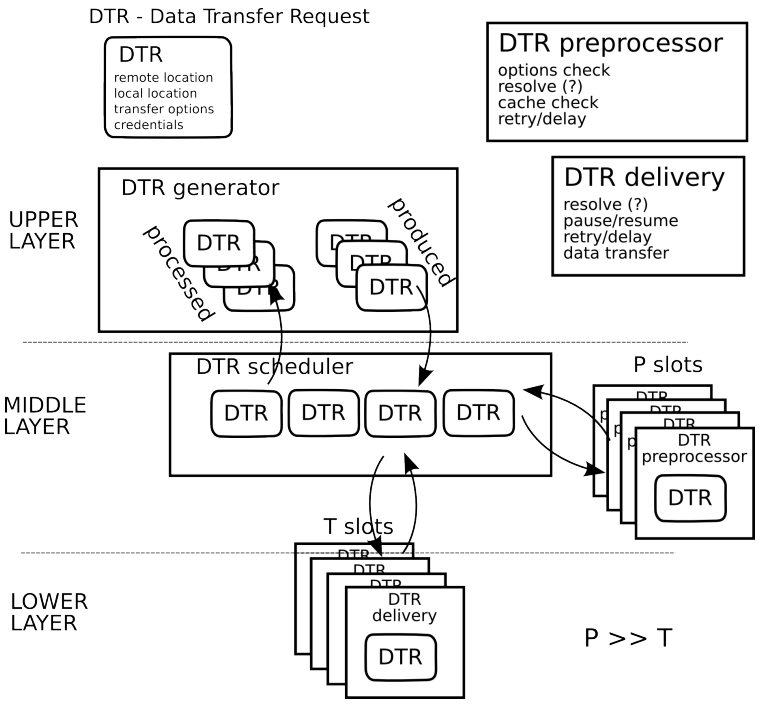


Figure 6.2: The architecture of the new data stating framework

section overrides any values set in `[grid-manager]`. As before, per-job logging information is in the `<control_dir>/job.id.errors` files.

Specific transfer shares are defined slightly differently from the old framework - rather than a set number of slots per share, a *priority* from 1 (lowest) to 100 (highest) is specified. The default priority for a share is 50. In the following example VO roles are used to assign shares, the atlas `slow_prod` role is assigned a low priority share and the atlas `validation` role is assigned a higher priority share:

```
sharetype="voms:role"
definedshare="atlas:slow-prod 20"
definedshare="atlas:validation 90"
```

Example: a user wants their job to be high (but not top) priority and specifies ("priority"= "80") in the job description. The user has a VOMS proxy with no role defined and submits the job to a site with the above configuration. The job is assigned to the default share and DTRs have priority 40 (50 x 80 / 100). The user then creates a VOMS proxy with the ATLAS validation role and submits the same job to the same site. This time the job goes to the configured atlas:validation share and the DTRs have priority 72 (90 x 80 / 100). The priority of a share determines how many slots it can use when multiple shares are competing for a limited number of physical transfer slots. Within a share, the priority of each DTR determines the order in which they are processed.

For more details please see the data staging page of the NorduGrid wiki[‡].

6.4 Job states

Figure 6.3 shows the *internal* states a job goes through, also listed here:

- **Accepted**: the job has been submitted to the CE but hasn't been processed yet.
- **Preparing**: the input data is being gathered.
- **Submitting**: the job is being submitted to the local resource management system (LRMS).

[‡] `http://wiki.nordugrid.org/index.php/Data_Staging`

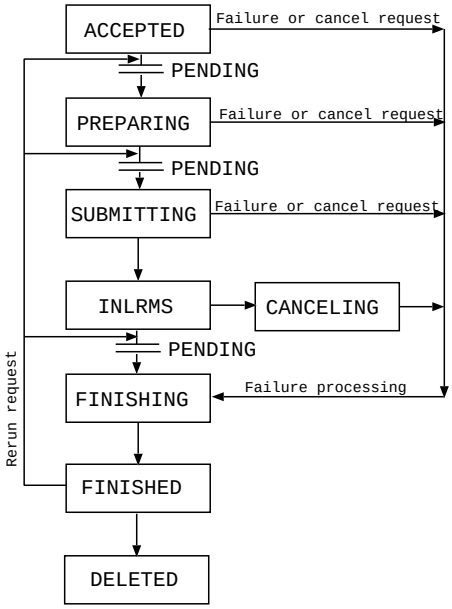


Figure 6.3: Job states

- **Executing (InLRMS)**: the job is queued or being executed in the LRMS.
- **Killing (Canceling)**: the job is being canceled.
- **Finishing**: the output data is being processed (even if there was a failure).
- **Finished**: the job is in this state either it finished successfully or there was an error during one of the earlier steps.
- **Deleted**: after specified amount of days the job gets deleted and only minimal information is kept about it.

Limits can be configured on the CE for the number of jobs in some states. If the limit reached, new jobs would stay in the preceding state (indicated by the **Pending** prefix). It is possible to re-run a job which is in the **Finished** state because of a failure. In this case the job would go back to the state where the failure happened.

These are internal states which are translated into more user-friendly external states when presented to the users. These external states take into account additional information, not just the internal state, so one internal state can correspond to multiple external states. In this list every row starts with the internal state followed by a colon and then the possible external states:

- ACCEPTED: ACCEPTING, ACCEPTED
- PREPARING: ACCEPTED, PREPARING, PREPARED
- SUBMITTING: PREPARED, SUBMITTING
- INLRMS: INLRMS, EXECUTED
- FINISHING: EXECUTED, FINISHING
- FINISHED: FINISHED, FAILED, KILLED
- CANCELING: KILLING
- DELETED: DELETED

6.5 Cache

6.5.1 Structure of the cache directory

Cached files are stored in sub-directories under the *data* directory in each main cache directory. Filenames are constructed from an SHA-1 hash of the URL of the file and split into subdirectories based on the two initial characters of the hash. In the extremely unlikely event of a collision between two URLs having the same SHA-1 hash, caching will not be used for the second file.

When multiple caches are used, a new cache file goes to a randomly selected cache, where each cache is weighted according to the size of the file system on which it is located. For example: if there are two caches of 1TB and 9TB then on average 10% of input files will go to the first cache and 90% will go to the second cache.

Some associated metadata including the corresponding URL and an expiry time, if available, are stored in a file with the same name as the cache file, with a *.meta* suffix.

For example, with a cache directory */cache*, the file

lfc://atlaslfc.nordugrid.org/grid/atlas/file1
is mapped to
/cache/data/78/f607405ab1df6b647fac7aa97dfb6089c19fb3

and the file */cache/data/78/f607405ab1df6b647fac7aa97dfb6089c19fb3.meta* contains the original URL and an expiry time if one is available.

At the start of a file download, the cache file is locked, so that it cannot be deleted and so that another download process cannot write the same file simultaneously. This is done by creating a file with the same name as the cache filename but with a *.lock* suffix. This file contains the process ID of the process and the hostname of the host holding the lock. If this file is present, another process cannot do anything with the cache file and must wait until the cache file is unlocked (i.e. the *.lock* file no longer exists). The lock is continually updated during the transfer, and is considered stale if 15 minutes have passed since the last update. These stale locks, caused for example by a download process exiting abnormally, will therefore automatically be cleaned up. Also, if the process corresponding to the process ID stored inside the lock is no longer running on the host specified in the lock, it is safe to assume that the lock file can be deleted. If a file is requested which already exists in the cache (and is not locked), the cache file is not locked, but checks are done at the end of cache processing to ensure the file was not modified during the processing.

6.5.2 How the cache works

If a job requests an input file which can be cached or is allowed to be cached, it is stored in the selected cache directory, then a hard link is created in a per-job directory, under the *joblinks* subdirectory of the main cache directory. Then depending on the configuration, either the hard-link is copied or soft-linked to the SD. The former option is advised if the cache is on a file system which will suffer poor performance from a large number of jobs reading files on it, or the file system containing the cache is not accessible from worker nodes. The latter option is the default option. Files marked as executable in the job will be stored in the cache without executable permissions, but they will be copied to the SD and the appropriate permissions applied to the copy.

The per-job directory is only readable by the local user running the job, and the cache directory is readable only by the A-REX user. This means that the local user cannot access any other users' cache files. It also means that cache files can be removed without needing to know whether they are in use by a currently running job. However, as deleting a file which has hard links does not free space on the disk, cache files are not deleted until all per-job hard links are deleted. **IMPORTANT:** If a cache is mounted from an NFS server and the A-REX is run by the root user, the server must have the *no_root_squash* option set for the A-REX host in the */etc/exports* file, otherwise the A-REX will not be able to create the required directories. Note that when running A-REX under a non-privileged user account, all cache files will be owned and accessible by the same user, and therefore modifyable by running jobs. This is potentially dangerous and so cacheing should be used with caution in this case.

If the file system containing the cache is full and it is impossible to free any space, the download fails and is retried without using cacheing.

Before giving access to a file already in the cache, the A-REX contacts the initial file source to check if the user has read permission on the file. In order to prevent repeated checks on source files, this authentication information is cached for a limited time. On passing the check for a cached file, the user's DN is stored in the *.meta* file, with an expiry time equivalent to the lifetime remaining for the user's proxy certificate. This means that the permission check is not performed for this user for this file until this time is up (usually several hours). File creation and validity times from the original source are also checked to make sure the cached file is fresh enough. If the modification time of the source is later than that of the cached file, the file will be downloaded again. The file will also be downloaded again if the modification date of the source is not available, as it is assumed the cache file is out of date. These checks are not performed if the DN is cached and is still valid.

The A-REX checks the cache periodically if it is configured to do automatic cleaning. If the used space on the file system containing the cache exceeds the high water-mark given in the configuration file it tries to remove the least-recently accessed files to reduce size to the low water-mark.

6.5.3 Remote caches

If a site has multiple A-REXs running, an A-REX can be configured to have its own caches and have read-only access to caches under the control of other A-REXs (remote caches). An efficient way to reduce network traffic within a site is to configure A-REXs to be under control of caches on their local disks and have caches on other hosts as remote caches. If an A-REX wishes to cache a file and it is not available on the local cache, it searches for the file in remote caches. If the file is found in a remote cache, the actions the A-REX takes depends on the policy for the remote cache. The file may be replicated to the local cache to decrease the load on the remote file system caused by many jobs accessing the file. However, this will decrease the total number of cache files that can be stored. The other policy is to use the file in the remote cache, creating a per-job directory for the hard link in the remote cache and bypassing the local cache completely. The usual permission and validity checks are performed for the remote file. Note that no creation or deletion of remote cache data is done - cache cleaning is only performed on local caches.

6.5.4 Cache cleaning

The cache is cleaned automatically periodically (every 5 minutes) by the A-REX to keep the size of each cache within the configured limits. Files are removed from the cache if the total size of the cache is greater than the configured limit. Files which are not locked are removed in order of access time, starting with the earliest, until the size is lower than the configured lower limit. If the lower limit cannot be reached (because too many files are locked, or other files outside the cache are taking up space on the file system), the cleaning will stop before the lower limit is reached.

Since the limits on cache size are given as a percentage of space used on the filesystem on which the cache is located, it is recommended that each cache has its own dedicated file system. If the cache shares space with other data on a file system, changes in the amount of non-cache data will result in changes in the available cache space.

With large caches mounted over NFS and an A-REX heavily loaded with data transfer processes, cache cleaning can become slow, leading to caches filling up beyond their configured limits. For performance reasons it may be advantageous to disable cache cleaning by the A-REX, and run the *cache-clean* tool independently on the machine hosting the file system. Please refer to Section **5.1.5**, *Cache administration*.

Caches can be added to and removed from the configuration as required without affecting any cached data, but after changing the configuration file, the A-REX should be restarted. If a cache is to be removed and all data erased, it is recommended that the cache be put in a *draining* state until all currently running jobs possibly accessing files in this cache have finished. In this state the cache will not be used by any new jobs, but the hard links in the *joblinks* directory will be cleaned up as each job finishes. Once this directory is empty it is safe to delete the entire cache

6.6 Transfer shares

For many jobs, large amounts of input and output data can mean significant time is spent in the PREPARING and FINISHING states gathering input data and writing output data. With FIFO processing, this can lead to one user or group of users blocking the queue for others. The A-REX implements a sharing system to avoid this problem, by assigning each user or group of users to a "transfer share" and specifying a limit on the number of data transfer processes per share. If one user's jobs' transfer share is using the maximum number of processes and another user submits jobs which are assigned to a different share, the second user's jobs can immediately go to PREPARING, up to the same maximum limit of processes. This means that no matter how many jobs the first user submits, the second user's jobs are not blocked. Assuming the bandwidth from the sources of input data for both users' jobs is similar, the available throughput will then be split evenly between the two users' jobs.

If a limit on the total number of data transfer processes is set in the *maxload* option, the maximum number of processes per transfer share is set by splitting the total maximum evenly among all the shares with jobs in data transfer states, up to the maximum allowed per share.

The scheme used to assign jobs to transfer shares can be set in the *maxloadshare* option. Possible values are:

- *dn* - each job is assigned to a share based on the DN of the user sumbitting the job.
- *voms:vo* - if the user's proxy is a VOMS [**?**] proxy the job is assigned to a share based on the VO specified in the proxy. If the proxy is not a VOMS proxy a default share is used.
- *voms:role* - if the user's proxy is a VOMS proxy the job is assigned to a share based on the role specified in the first attribute found in the proxy. If the proxy is not a VOMS proxy a default share is used.
- *voms:group* - if the user's proxy is a VOMS proxy the job is assigned to a share based on the group specified in the first attribute found in the proxy. If the proxy is not a VOMS proxy a default share is used.

It's possible to distinguish some transfer shares and assign them a limit different from what's specified in *maxloadshare*. It's done by *share_limit* option. *share_limit* can only be used if *maxloadshare* has been already set before. Depending on the sharing mechanism used by *maxloadshare*, the proper name for the share should be specified, as illustrated by the following examples (note, that in dn case spaces are allowed, the configuration parser will take care of them):

- *dn*: /O=Grid/O=NorduGrid/OU=domainname.com/CN=Jane Doe
- *voms:vo*: voname
- *voms:role*: voname:rolename
- *voms:group*: /voname/groupname

The specific shares, specified in *share_limit*, are processed differently from the other shares. A-REX reserves an indicated number of processes for each specific share. The number of unreserved processes is then split evenly between the ordinary shares, as determined by *maxloadshare*. So the specific shares have a strict, non-decreaseable limit, unlike all the ordinary shares, whose limit can be decreased while A-REX tries to split the load evenly. However, A-REX reserves processes only for active specific shares, i.e. shares to which at least one active job on the resource belongs to. If the share is not active, its slots are used in overall splitting between ordinary transfer shares.

A particular case is when A-REX reserves more processes than specified in *maxjobs*. A-REX will process jobs from specific share at FIFO-basis and stop at reaching *maxjobs* number of processes, even if some specific shares haven't reached their limits. Also in this situation each ordinary share is allowed to launch only one upload and download process.

If VOMS is not supported, the *dn* scheme is the only option that should be used, as using a VOMS-based scheme will lead to all jobs being assigned to the default share. The current number of jobs processing and pending processing for each share can be seen with the command *gm-jobs -s*.

Important: If a sharing mechanism based on VOMS is used, server certificates for each supported VO must be installed. It is possible to either download the public key of each VOMS server, or create a special file for each VO containing the server's DN and its CA DN. Instructions are given on NorduGrid's web site at `http://www.nordugrid.org/documents/voms-notes.html`.

When XML file only is used to configure the A-REX, the transfer shares can be implemented by defining *maxLoadShare* (the limit itself) and *loadShareType* (the scheme used) elements inside *loadLimits* block. For defining the specific shares, *shareLimit* sub-blocks with *name* and *limit* elements can be used after *maxLoadShare*.

6.7 Batch system back-ends implementation details

The batch system back-ends are what tie the ARC Grid middleware (through the ARC Resource-coupled EXecution service, A-REX to the underlying cluster management system or LRMS. The back-ends consist of set of a shell and Perl scripts whose role are twofold:

1. to allow A-REX, to control jobs in the LRMS including job submit, status querying and cancel operations.

2. to collect information about jobs, users, the batch system and the cluster itself for the Information System.

The job control part of the LRMS interface is handled by the A-REX. It takes care of preparing a native batch system submission script, managing the actual submission of the batch system job, cancellation of job on request and scanning for completed batch jobs. Besides the LRMS job control interface it is also A-REX which provides e.g. the data staging and communication with the Grid client, provides RTE environments, arranges file staging (to the node via LRMS capability), dealing with stdout/stderr, etc. The job control batch system interface of A-REX requires three programs. These programs can be implemented any way the designer sees it fits, but all the existing back-end interfaces use shell scripting for portability and ease of tailoring to a specific site. A-REX will call the following programs: cancel-LRMS-job, submit-LRMS-job, and scan-LRMS-job where LRMS is replaced with the short hand name for the LRMS; e.g. cancel-pbs-job. The scripts are described one by one in the following subsections.

6.7.1 Submit-LRMS-job

The submit program is the most involved. It is called by A-REX once a new job arrives and needs to be submitted to the LRMS. It is given the GRAMi file as argument on execution. The GRAMi file is a file in the job control directory containing the job description in a flat list of key-value pairs. This file is created by A-REX and is based on the JSDL job description. Submit-LRMS-job then has to set up the session directories, run-time environment and anything else needed. Then it submits the job to the local LRMS. This is normally done by generating a native job script for the LRMS and then running the local submit command, but it can also be done through an API if the LRMS supports it.

6.7.2 Cancel-LRMS-job

If a Grid user cancels his job, the message will reach the grid-manager. The manager will then call the cancel-LRMS-job for the suitable back-end. The cancel script is called with the GRAMi file containing information about the job such as the job id in the LRMS. Cancel-LRMS-job must then use that information to find the job and remove it from the queue or actually cancel it if it is running in the LRMS.

6.7.3 Scan-LRMS-job

The scan-LRMS-job is run periodically. Its job is to scan the LRMS for jobs that have finished. Once it has found a finished job it will write the exit-code of that job to the file job.{gridid}.lrms_done in the

ARC job status directory[§]. Then it will call the gm-kick program to notify A-REX about the finished job. Subsequently, A-REX starts finalizing the job.

Generally, two approaches are taken to find jobs which are finished in LRMS. One is to directly ask the LRMS. Since all started Grid jobs have its own status file[¶] found in the job status directory, this can be done by checking if the status is "INLRMS" in this file. If so, a call to the LRMS is made asking for the status of the job (or jobs if several jobs have status "INLRMS"). If it is finished, it is marked as such in the job status directory, and the gm-kick program is activated. For most LRMSs the information about finished jobs are only available for a short period of time after the job finished. Therefore appropriate steps have to be taken if the job has the status "INLRMS" in the job status directory, but is no longer present in the LRMS. The normal approach is to analyze the job's status output in the session directory.

The second approach is to parse the LRMSs log files. This method has some drawbacks like e.g.: A-REX has to be allowed read access to the logs. The back-end will then have to remember where in the log it was last time it ran. This information will have to be stored in a file somewhere on the front-end.

6.7.4 PBS

The job control batch interface makes use of the `qsub` command to submit native PBS job scripts to the batch system. The following options are used:

`-l nodes, cput, walltime, pvmem, pmem,`

`-W stagein, stageout`

`-e, -j eo`

`-q`

`-A`

`-N`

For job cancellation the `qdel` command is used. To find completed jobs, i.e. to scan for finished jobs the `qstat` command or the `PBS server log file` is used.

The information system interface utilizes the `qstat -f -Q queuename` and `qstat -f queuename` commands to obtain detailed job and queue information. `qmgr -c "list server"` is used to determine PBS flavour and version. The `pbsnodes` command is used to calculate total/used/free cpus within the cluster. In case of a MAUI scheduler the `showbf` command is used to determine user freecpu values. All these external PBS commands are interfaced via parsing the commands' output.

6.7.5 Condor

The job control part of the interface uses the `condor_submit` command to submit jobs. Some of the options used in the job's ClassAd are:

`Requirements` – is used to select the nodes that may run the job. This is how ARC queues are implemented for Condor.

`Periodic_remove` – is used to enforce cputime and walltime limits.

`Log` – the job's condor log file is parsed by the information scripts to find out whether the job was suspended.

The information system component uses the following Condor commands:

`condor_status -long` – for collecting information about nodes

`condor_status -format "%s\n" Machine -constraint '...'` – for listing nodes that make up an ARC queue.

[§]normally /var/spool/nordugrid/jobstatus/, but can be set via the controldir variable of arc.conf
[¶]job.{gridid}.status

`condor_q -long -global` – for monitoring running jobs.

`condor_history -l jobid` – for collecting information about finished jobs. Further cues are taken from the job's condor log file and the body of the email sent by Condor when a job completes.

6.7.6 LoadLeveler

The LoadLeveler back-end uses LoadLeveler's command line interface(CLI) commands to submit and cancel jobs. All information in the information system is similarly parsed from the output of CLI commands. It does not parse any log files, nor does it use the binary APIs. The reason that the back-end is completely based on the CLI is that the log files are normally kept on another machine than the front end and that the binary API for LL changes quite often. Often with each new version of LL.

6.7.7 Fork

The Fork back-end implements an interface to the "fork" UNIX command which is not a batch system. Therefore the back-end should rather be seen as an interface to the operating system itself. Most of the "batch system values" are determined from the operating system (e.g. cpu load) or manually set in the configuration file.

6.7.8 LSF

The LSF implementation of the back-end are based solely on parsing and running LSF's command line interface commands. No log files or other methods are used. To get the correct output o any output at all the back-end needs to have an apropriate LSF profile. The path to this profile must be set in arc.conf. It will then be executed by the back-end before running any of LSF's CLI commands.

6.7.9 SGE

The SGE back-end's commands are similar to the PBS commands. These commands are used in the code:

Submit job:

- qsub -S /bin/sh (specifies the interpreting shell for the job)

Get jobs status:

If the job state is not suspended, running or pending then its state is failed.

- qstat -u '*' -s rs (show the running and suspended jobs status)
- qstat -u '*' -s p (show the pending jobs status)
- qstat -j job_id (long job information)
- qacct -j job_id (finished job report)

Job terminating:

- qdel job_id (delete Sun Grid Engine job from the queue)

Queue commands:

- qconf -spl (show a list of all currently defined parallel environments)
- qconf -sql (show a list of all queues)
- qconf -sep (show a list of all licensed processors/slots)

- qstat -g c (display cluster queue summary)
- qconf -sconf global (show global configuration)
- qconf -sq queue_name (show the given queue configuration)

Other:

- qstat -help (show Sun Grid Engine's version and type)

6.8 Clustering A-REX

For large clusters, using a single machine for all input and output file transfer, as well for the interaction with the LRMS and Information System, can limit the job throughput of the cluster. Running several A-REXes on separate hosts can help spread the hardware and network load. A single GFS can feed jobs to several A-REXes, hence a cluster with many A-REXes still appears as a single site to the outside world. When a job is submitted, the GFS jobplugin assigns a random control directory to use for the job from the main *controldir* specified in the A-REX configuration and any extra *remotegmdirs* specified in the jobplugin configuration. Note that it is not necessary to run an A-REX on the GFS host, in other words a configuration with only *remotegmdirs* is possible. Also note that functionality of clustered A-REXes is not (yet) possible with WS interface because each A-REX has own integrated job vcontrol interface. In that case clustering is achieved by using distributed data staging service as described in Section 6.3.

Each control directory is used by a separate A-REX, therefore for every *remotegmdirs* command in the GFS configuration, there must be a A-REX running which defines the corresponding *controldir* and *sessiondir* in its configuration file. Each A-REX can run independently on its own host, the only requirement is that the control and session directories must be accessible on the GFS host, and the GFS user must have write access to these directories. It is recommended that these directories are local to the GFS host and exported (via NFS for example) to the other hosts, rather than being on a remote filesystem and exported to the GFS host. This means that any glitches in the network do not cause the GFS host to hang. It is also important that the local user accounts on each host must be synchronised with the GFS host. Any A-REX is not aware that any other A-REXes are running, as they only see what the GFS decides should go into their own control directory. All communication between the GFS and a A-REX is through the A-REX's control directory. Note that in remote control directories there is no way to specify control directories per user, as with the *control* command. Only the *controldir* command can be used and all users will use the same control directory.

One feature of this design is that multiple A-REXes can share the same LRMS, and hence compete with each other to submit jobs. Therefore any LRMS settings in the configuration files must be carefully matched in order not to bias one A-REX over another. In most cases each host's configuration file can be identical apart from the control and session directories. Some configuration sections such as the GFS and infosys sections will be ignored by the remote A-REX hosts as these services are not running.

Cacheing can be set up in a variety of different ways. Each A-REX can have its own cache, completely independent from any other, which will lead to popular files being replicated in many caches. Or, all caches can be shared with all A-REXes, which means no replication between caches but heavy intra-site network traffic if the cache file systems are hosted on different hosts. Another option is to give each A-REX its own cache, but access to the other caches as remote caches. This avoids replicating files and intra-site network traffic. Replication can still be enabled by specifying "replicate" as the link_path for remote cache dirs, and the advantage of this is that files are copied from the remote cache rather than being downloaded again from source.

When setting up an extra A-REX, it is important that no other services (GFS, infosys) run on the host. The instances of these services running on the "main" host take care of all the A-REXes. It is recommended therefore that packages containing *gridftpd* and *grid-infosys* are not installed on these hosts, and if they are, care should be taken to remove or disable scripts which could be started automatically (usually in *$ARC_LOCATION/etc/init.d/*). No host certificates are required for hosts which only run a A-REX instance but CA certificates are needed for contacting remote services when downloading and uploading job input and output files. Note that in this multiple A-REX set up, there is a one to one relationship between control and session directories, hence multiple *sessiondir* commands cannot be used in a A-REX configuration.

If no A-REX service is run on the GFS host then the option *infosys_compat="enable"* must be set in the *[infosys]* section of the GFS host configuration (the infosys and GFS must be on the same host). This means the legacy infoproviders in the infosys are used instead of those in A-REX. If this option is not set then the A-REX infoproviders must be used and A-REX started on the GFS host. However it is possible to start A-REX without any (local) control directories defined - the grid-manager thread for processing jobs will exit straight away but the thread running the infoproviders will continue and will collect information from all the remote A-REXes.

When adding a new *remotegmdirs* option to the configuration, there is no need to restart the GFS, as the configuration is read dynamically by the job plugin upon job submission. However the A-REX serving those new directories must be started before adding them to the GFS configuration. If an A-REX is to be removed from the system, it is best to set the *remotegmdirs* to a draining state (for how see description of this parameter in configuration section), so that running jobs are still accessible but no new jobs will be assigned there. When the A-REX has completed all jobs and all job output has been retrieved (for example there are no jobs found when running gm-jobs), it is safe to remove from the GFS configuration.

The status of all A-REXes is monitored by the information system through heartbeat files in each control directory. If all A-REXes are running ok then jobs can be submitted to the site. If some A-REXes are up and some are down then the site will publish a "degraded" state and no new jobs can be submitted. If all A-REXes are down then the site publishes a "critical" state and no new jobs can be submitted.

6.9 JURA: The Job Usage Reporter for ARC

6.9.1 Overview

JURA is a stand-alone binary application which is periodically run by the A-REX (see Figure 6.5). There is no designated configuration file for JURA, nor is the configuration file of A-REX read directly by the application. Instead, options related to reporting are included within the job log files generated by the A-REX or supplied via command line argument. The primary purpose of these job log files is to hold metadata about jobs starting, running and stopping. This is the main input of JURA.

The application is run periodically. First, it processes the job log files, and based on the target accounting service specified in them, JURA creates usage records in a format appropriate for the target accounting service. Then these records are sent to one or more accounting services, referred to as *reporting destinations* in this document. Several reporting destinations are supported, these can be configured by the system administrator in the A-REX configuration file, and in addition, the user submitting the job can specify destinations in the job description.

About configuration of JURA, see **4.4.5**, *Accounting with JURA*.

6.9.2 Job log files

The A-REX puts the following extra information into the job log files:

- ***key_path*** – Path to the private key file used when submitting records.
- ***certificate_path*** – Path to the certificate file used when submitting records.
- ***ca_certificates_dir*** – Directory holding the certificates of trusted CAs.
- ***accounting_options*** – Additional configuration options for JURA.

The A-REX generates at least two job log files for each job and for each reporting destination: one at the time of job submission, another one after the job finishes, and possibly others at each start and stop event.

The job log files generated by A-REX reside under the directory `<control_dir>/logs`. The name of the job log files consist of the ID of the job and a random string to avoid collision of multiple job log files for the same job: `<jobid>.<random>`.

The job log file consists of "`name=value`" lines, where "`value`" is either a job-related resource usage data or a configuration parameter.

A-REX log entry	OGF-UR	CAR	ARC-UR	APEL-UR
nodename, ngjobid (hyphen-separated, with custom prefix)	RecordIdentity:*	RecordIdentity*		
nodename		Host		
globalid	GlobalJobId:	GlobalJobId	globaljobid	
localjobid	LocalJobId:	LocalJobId*	localjobid	LocalJobId
[MISSING]	ProcessId:		processid	
usersn	GlobalUserName:	GlobalUserName	globaluserid	GlobalUserName
localuser	LocalUserId:	LocalUserId*	localuserid	LocalUserId
jobname	JobName:	JobName	jobname	
[MISSING]	Charge:	Charge	charge	
status (no conversion yet!)	Status:*	Status*	status	
usedwalltime (sec)	WallDuration: (ISO)	WallDuration* (ISO)	usedwalltime	WallDuration
usedcputime (sec)	CpuDuration: (ISO)	CpuDuration* (ISO) all	usedcputime	CpuDuration
usedusercputime (sec)	user	user		
usedkernelcputime (sec)	kernel	kernel		
submissiontime	StartTime: (UTC)	StartTime* (UTC)	submissiontime	StartTime
endtime	EndTime: (UTC)	EndTime* (UTC)	endtime	EndTime
nodename	MachineName:		nodename	Site
nodename	Host:			
clienthost (port number removed)	SubmitHost:		submithost	
headnode		MachineName, SubmitHost*, Site*		SubmitHost
lrms		Infrastructure* (desc.)		
queue	Queue:	Queue*	queue (lrms)	
projectname	ProjectName:	GroupAttribute	projectname	
[MISSING]		Group		
exitcode		ExitStatus		
[MISSING]		ServiceLevel*		
[MISSING]	Network:		network	
[MISSING]	Disk:		useddisk	
usedmemory	Memory: virtual,average	Memory: Share, average	usedmemory	MemoryReal
usedmaxresident	Memory: physical,max	Memory: Physical, max		
usedaverageresident	Memory: physical,average	Memory: Physical, average		
[MISSING]	Swap:	Swap	usedswap	
nodecount	NodeCount:	NodeCount	nodecount	NodeCount
[MISSING]	Processors:	Processors	processors	Processors
[MISSING]	TimeDuration:			
[MISSING]	TimeInstant:	TimeInstant		
[MISSING]	ServiceLevel:			
[MISSING]	Extension:			
[MISSING]				FQAN
[MISSING]				ServiceLevel
[MISSING]				ServiceLevelType

Figure 6.4: Mapping of usage record properties, mandatory properties are flagged with an asterisk (*)

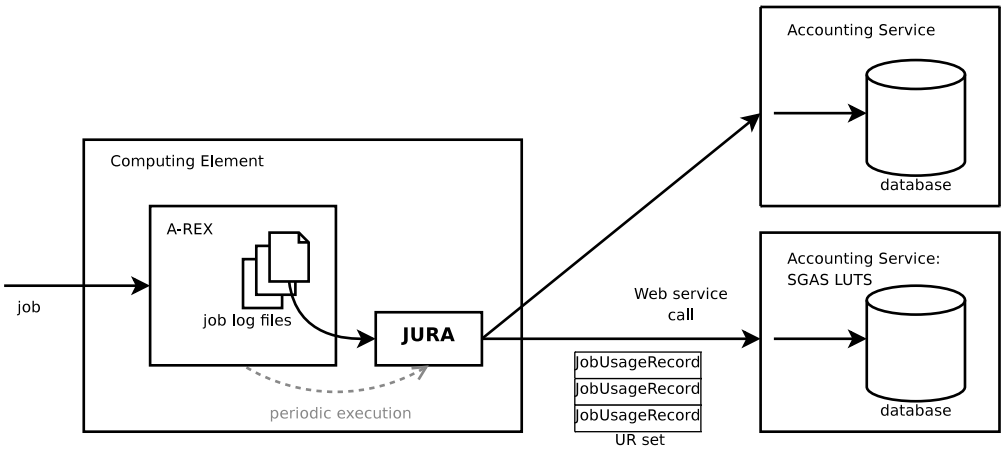


Figure 6.5: The usage reporting mechanism

If interactive mode is not activated by the "`-u`" option, after successful submission to a reporting destination, the job log file is deleted, thus preventing multiple insertion of usage records. If submission fails, the log files are kept, so another attempt is made upon a subsequent run of JURA. This mechanism will be repeated until the expiration time passes at which point the next execution of JURA removes the file without processing.

6.9.3 Archiving

The archiving functionality allows to store generated usage records in a specified directory on the disk. If enabled, the generated usage records are written to files named "`usagerecord.<jobid>.<random>`". If a job log file is processed repeatedly – for example because of temporary connection failures to the remote accounting service – and a respective usage record archive file already exists, then the usage record is not generated again. Instead, the contents of the archive file are used without change (the creation time stamp is also retained).

6.9.4 Security

The JURA executable runs with the same user privileges as the A-REX. The owner of a job log file is the local user mapped for the submitter entity of the corresponding job. Since these files contain confidential data, A-REX restricts access to them allowing only read access for the job owner, thus when JURA is executed by A-REX it is allowed to read and delete job log files.

All usage records are submitted using the X.509 credentials specified by the value of the `jobreport_credentials` value of the A-REX configuration file. No proxies are used.

6.9.5 Mapping of job log entries to usage record properties

See **6.4**, *Mapping of usage record properties, mandatory properties are flagged with an asterisk (*).*

6.10 The XML and the INI configuration formats

This section clarifies the roles of the different configuration file formats used in ARC.

The main service of the ARC Computing Element is the A-REX, which runs in a *service container* called the *HED*. The HED is part of the ARC middlware, it contains a daemon and several loadable modules. The configuration of the HED can be done with an XML configuration file, which describes how to connect the several internal components to eachother.

Administrators of the ARC CE usually only uses the `arc.conf` configuration file. The HED service container does not understand the format of the `arc.conf` file, that's why the init script of the A-REX has to parse the `arc.conf` and generate a configuration file which is suitable for the HED service container.

The low-level configuration format of the HED service container is an XML-based format, which allows for very fine-grained configuration of all the internal components and other services than the A-REX.

There is a higher-level configuration possibility, which has the INI format, and which is transformed into the XML configuration when the HED service container is started. Putting configuration options of the INI config enables and sets sections of an XML configuration profile.

Although the original `arc.conf` also has an INI-like format, it should not be confused with the high-level INI configuration of the HED service container.

To summarize the three configuration formats:

arc.conf parsed by the A-REX init script and the A-REX itself. The init script generates an XML configuration from it to configure the HED service container

XML is the low-level configuration format of the HED service container

INI is the high-level configuration format of the HED service container (and it has nothing to do with the `arc.conf`)

For details on the XML and INI configuration, please see [**?**].

6.11 The internals of the service container of ARC (the HED)

This section describes the internal components of the ARC service container.

The main service of the ARC Computing Element is the A-REX, which runs in a *service container* called the *HED*. The HED is part of the ARC middlware, it contains a daemon and several loadable modules. The configuration of the HED can be done with an XML configuration file, which describes how to connect the several internal components to eachother. Here follows a short description of these internal components.

6.11.1 The MCCs

The HED service container has zero or more *message chains*. A message chain can contain *Message Chain Components (MCCs)*, *Plexers* and *Services*. These components are passing messages to eachother, which messages can have various extra attributes which also can be read and set by these components.

An MCC gets a message from a previous MCC, does something with it, passes it to the next MCC (or Plexer or Service), then "waits" for the response of this next MCC, then it does something with the response and passes it back to the previous MCC. (In case of the server-side TCP MCC, it is listening on a network port, and gets the message from there, not from another MCC, and the response will be sent back to the network, not to another MCC—the client-side TCP MCC is not listening but opening a connection to a host and port and sending the message into it.)

The Plexer checks the path of the destination of the message, and based on the matching of regular expressions it sends the message to one of the configured Services (or other MCCs or Plexers). The path checked by the Plexer comes from a message attribute called "ENDPOINT". The Plexer treats this as a URL, and uses the "path" part of it. This message attribute is set by the MCCs before the Plexer, usually by the HTTP MCC, which puts the URL of the HTTP request there. (But it is possible that a SOAP request contains a WS-Addressing information with an endpoint in it, then this will be set as "ENDPOINT" by the SOAP MCC.)

A Service processes the message and produces a result which it will pass back to the previous element (Plexer, MCC). If we only have one web service in a chain which lives at the root HTTP path (e.g. https://hostname:port), then we don't need a Plexer. If we want to run a Service without any interface, then we don't need MCCs, we can have a chain with a single Service inside.

The XML configuration of the HED has "Chain" XML elements which contains the MCCs (in XML elements called "Component") and Plexers ("Plexer") and Services ("Service").

The MCCs:

TCP handles the external network communication (IPv4 and IPv6)

TLS handles the TLS/SSL connection, it is usually directly after the TCP MCC, it extracts information from the message like the Subject Name (DN) of the client

HTTP handles HTTP communication, should be directly after the TLS MCC (or can be directly after the TCP MCC, if we want HTTP without TLS security). The HTTP MCC can route the message to different components depending on the HTTP method (POST, GET, PUT, HEAD). Usually the POST method goes through the SOAP MCC, while the others skip the SOAP handling.

SOAP handles the SOAP communication. To use it, the HTTP MCC should directly route the POST message to the SOAP MCC.

If there is a Plexer, a useful configuration could be the following: the HTTP MCC sends the POST messages through the SOAP MCC to the Plexer, and sends the GET, PUT and HEAD messages directly to the Plexer. Then there are several Services sitting after the Plexer, and the Plexer would route the message to a selected service based on the configured regular expressions. The Plexer can handle messages from the HTTP MCC and from the SOAP MCC as well (actually, from the TCP MCC also, but that's currently not useful at all, because there is no endpoint path in that case). This enables services like the A-REX to handle both SOAP messages and non-SOAP requests (e.g. uploading input files through HTTP).

6.11.2 The SecHandlers

All MCCs and Services can have zero or more *Security Handlers (SecHandlers)* in one or more queues. Usually an MCC (or a Service) has two queues, one for the incoming message, and the other for the outgoing response. When the message comes into an MCC, the first SecHandler in the incoming queue gets it, and checks it. If it says "denied" then the message will be discarded and an error message will be sent back as a respone. If it says "ok" then the next SecHandler in the queue gets it, etc.

The SecHandlers adds security attributes to the message, which can be used later by other SecHandlers or Services (e.g. the A-REX can use them to figure out which user to map to).

In the XML configuration of the HED each MCC or Service element can have zero or more SecHandler elements, which specifies the name of the SecHandler (based on this name, the binary plugin will be loaded) and the name of the queue (called "event": incoming or outgoing).

ARC has the following SecHandlers:

- ArgusPEP (`arguspep.map`)
- IdentityMap (`identity.map`)
- DelegationCollector (`delegation.collector`)
- ArcAuthZ (`arc.authz`)
- DelegationSH (`delegation.handler`)
- LegacyMap (`arclegacy.map`)
- LegacySecHandler (`arclegacy.handler`)
- SAML2SSO_AssertionConsumerSH (`saml2ssoassertionconsumer.handler`)
- SAMLTokenSH (`samltoken.handler`)
- UsernameTokenSH (`usernametoken.handler`)

Each of them has an associated PluginDescriptor object, which are used the create the `.apd` files (Arc Plugin Description) which can be found next the the loadable modules (libraries on linux) in the installed location (e.g. `/usr/local/lib/arc`). The `.apd` files are generated by the `arcplugin` utility, and they are used by the HED when it tries to find the plugin based on its name. (If there are no `.apd` files, then all the modules have to be loaded in order to find the plugins, which takes more time.)

The names of the SecHandlers sometimes contains `.map` or `.handler` or `.authz` as a suffix. This is just a naming convention, they are all Security Handlers.

The most important SecHandlers:

IdentityMap

This security handler tries to map the Grid user (the DN which comes from the TLS MCC) to a local user. If it finds a local user, it puts the username into a security attribute which can be used later by other components. (This SecHandler never denies the message going forward, the worst thing can happen is that it doesn't find a local user, so the security attribute will be empty.)

It has three ways to do the mapping:

LocalName always maps to the specified local user regardless of the DN

LocalList uses the familiar grid-map file format to find the local name

LocalSimplePool maps to a pool of local user in a way that only one DN should be mapped to one local user (and a directory will contain the current mappings)

This SecHandler uses plugins called PDPs, see later.

ArcAuthZ

This is the main ARC security handler, which uses a couple of plugins (PDPs) to decide if a connection should go through or should be stopped and denied. (This does not do any Grid user – local user mapping.) So when a message arrives to the ArcAuthZ SecHandler, it will run the configured PDPs, and if any of them says "denied" then the message will be denied. If all the PDPs say "ok", then the message can go forward (this behaviour can be changed by configuration). About the PDPs, see later.

LegacyMap

This SecHandler does user mapping, it uses the `arc.conf` and does the mapping according to the `unixgroup`, `unixvo` and `unixmap` configuration parameters.

LegacySecHandler

This SecHandler does not map to local user or deny messages, only collects information which will be used later by the LegacyMap SecHandler (and by the Legacy PDP, see next section). It uses the `arc.conf`, and figures out which VOs and Groups the user belongs to.

6.11.3 The PDPs

Some of the SecHandlers use another layer of plugins, which are called *Policy Decision Points (PDPs).* (Currently only the IdentityMap and the ArcAuthZ SecHandlers use PDPs.) These SecHandlers have a queue of PDPs and they run them one by one, doing different things based on the results. The IdentityMap plugin has a given mapping policy (user, list, pool) for each PDP, and at the first PDP which returns "ok", it will stop running further PDPs and use the mapping policy configured for the given PDP. The ArcAuthZ runs all the PDPs, and only accepts the message if all of them returns "ok". (Although it can be configured differently, e.g. to accept the message if at least one PDP says "ok", or only accept a message if a PDPs says "deny", etc.)

The current PDPs:

- LegacyPDP (`arclegacy.pdp`)
- SimpleListPDP (`simplelist.pdp`)
- ArcPDP (`arc.pdp`)
- XACMLPDP (`xacml.pdp`)
- PDPServiceInvoker (`pdpservice.invoker`)
- DelegationPDP (`delegation.pdp`)
- AllowPDP (`allow.pdp`)
- DenyPDP (`deny.pdp`)

The most important ones:

LegacyPDP

This one check the previously set (by the LegacySecHandler) Group and VO attributes, and it also checks the `arc.conf`, and figures out if the given user is allowed or not.

SimpleListPDP

This one checks a given file with a list of DNs (can be a grid-map file), and only accepts messages from DNs listed in the file.

ArcPDP

This one parses policy file written in a general purpose policy language (developed by KnowARC) and makes a decision based on it.

AllowPDP

This one always allows.

DenyPDP

This one always denies.

6.12 How the a-rex init script configures the HED

The `a-rex` init script extracts information from the arc.conf, and creates an XML configuration for the HED. The A-REX service (living inside the HED) itself uses the `arc.conf` to configure itself, but there is a higher layer of configuration options which has to be set in the HED directly (e.g. authentication of the TLS communication), this configuration parameters has to be extracted from the `arc.conf` before the A-REX can even be started, and a proper XML configuration has to be assembled to configure the HED itself.

The `a-rex` init script first decides if the A-REX would have a web service interface or not. If the web service is disabled, then the XML configuartion of the HED would look like this:

```
<?xml version="1.0"?>
<ArcConfig
xmlns="http://www.nordugrid.org/schemas/ArcConfig/2007"
xmlns:arex="http://www.nordugrid.org/schemas/a-rex/Config">
<Server>
  <PidFile>$PID_FILE</PidFile>
  <Logger>
    <File>$LOGFILE</File>
    <Level>$LOGLEVEL</Level>
    <Backups>$LOGNUM</Backups>
    <Maxsize>$LOGSIZE</Maxsize>
    <Reopen>$LOGREOPEN</Reopen>
  </Logger>
</Server>
<ModuleManager>
  <Path>$ARC_LOCATION/@pkglibsubdir@/</Path>
</ModuleManager>
<Plugins><Name>arex</Name></Plugins>
<Chain>
  <Service name="a-rex" id="a-rex">
    <arex:gmconfig>$ARC_CONFIG</arex:gmconfig>
  </Service>
</Chain>
</ArcConfig>
```

The variables (names starting with a dollar sign) are substituted with values from the `arc.conf`. Here the message chain contains only a single A-REX service, which has one single config parameter: "gmconfig", which points to the location of the `arc.conf`. In this case the A-REX does not have any HTTP or SOAP interfaces, no SecHandlers, no PDPs, because everything is done by the GridFTP Server, which has a separate init script, it is a separate process, and it has all the authentication and authorization mechanisms built-in.

When the web service interface is enabled, then the job submission through the web service interface would go through through the following components:

- a TCP MCC listening on the given port:

  ```
  <Component name="tcp.service" id="tcp">
    <next id="tls"/>
    <tcp:Listen><tcp:Port>$arex_port</tcp:Port></tcp:Listen>
  </Component>
  ```

- a TLS MCC using the key and certificate and CA paths from the arc.conf, trusting all the VOMS servers, having a specific VOMSProcessing (relaxed, standard, strict, noerrors), having an IdentityMap SecHandler which uses the given gridmapfile to map the Grid users and maps to "nobody" in case of error, then having a LegacySecHandler which uses the arc.conf to match the client to groups and VOs configured there:

  ```
  <Component name="tls.service" id="tls">
    <next id="http"/>
    <KeyPath>$X509_USER_KEY</KeyPath>
    <CertificatePath>$X509_USER_CERT</CertificatePath>
    <CACertificatesDir>$X509_CERT_DIR</CACertificatesDir>
    <VOMSCertTrustDNChain>
      <VOMSCertTrustRegex>.*</VOMSCertTrustRegex>
    </VOMSCertTrustDNChain>
    <VOMSProcessing>$VOMS_PROCESSING</VOMSProcessing>
    <!-- Do initial identity mappping by gridmap file -->
  ```

```
  <SecHandler name="identity.map" id="map" event="incoming">
    <PDP name="allow.pdp"><LocalList>$GRIDMAP</LocalList></PDP>
    <PDP name="allow.pdp"><LocalName>nobody</LocalName></PDP>
  </SecHandler>
  <!-- Match client to legacy authorization groups -->
  <SecHandler name="arclegacy.handler" event="incoming">
    <ConfigFile>$ARC_CONFIG</ConfigFile>
  </SecHandler>
</Component>
```

- one HTTP MCC, one SOAP MCCs, and the Plexer, with POST messages going through SOAP to the Plexer, GET/PUT/HEAD messages going directly to the Plexer, which checks if the path is the configured `arex_path`, if yes, it sends the message to the A-REX, otherwise fails:

```
<Component name="http.service" id="http">
  <next id="soap">POST</next>
  <next id="plexer">GET</next>
  <next id="plexer">PUT</next>
  <next id="plexer">HEAD</next>
</Component>
<Component name="soap.service" id="soap">
  <next id="plexer"/>
</Component>
<Plexer name="plexer.service" id="plexer">
  <next id="a-rex">^/$arex_path</next>
</Plexer>
```

- then the A-REX itself, with ArcAuthZ SecHandler containing a single LegacyPDP which will decide based on the `[gridftpd/jobs]` section of `arc.conf` if this message can go through or should be denied, then a LegacyMap SecHandler which uses the `[gridftpd]` section of arc.conf to figure out which local user should the Grid user be mapped to, then the full URL of the A-REX is given to the service (which in theory could be figured out from the incoming messages, but it is safer to be set explicitly), then the location of the `arc.conf` is given to the service (otherwise it wouldn't know), then some extra limits are set:

```
<Service name="a-rex" id="a-rex">
  <!-- Do authorization in same way as jobs plugin of gridftpd does -->
  <!-- Beware of hardcoded block name -->
  <SecHandler name="arc.authz" event="incoming">
    <PDP name="arclegacy.pdp">
      <ConfigBlock>
        <ConfigFile>$ARC_CONFIG</ConfigFile>
        <BlockName>gridftpd/jobs</BlockName>
      </ConfigBlock>
    </PDP>
  </SecHandler>
  <!-- Perform client mapping according to rules of gridftpd -->
  <SecHandler name="arclegacy.map" event="incoming">
      <ConfigBlock>
        <ConfigFile>$ARC_CONFIG</ConfigFile>
        <BlockName>gridftpd</BlockName>
      </ConfigBlock>
  </SecHandler>
  <arex:endpoint>$arex_mount_point</arex:endpoint>
  <arex:gmconfig>$ARC_CONFIG</arex:gmconfig>
  <arex:InfosysInterfaceMaxClients>
    $MAX_INFOSYS_REQUESTS
  </arex:InfosysInterfaceMaxClients>
```

```
    <arex:JobControlInterfaceMaxClients>
      $MAX_JOB_CONTROL_REQUESTS
    </arex:JobControlInterfaceMaxClients>
    <arex:DataTransferInterfaceMaxClients>
      $MAX_DATA_TRANSFER_REQUESTS
    </arex:DataTransferInterfaceMaxClients>
  </Service>
```

In summary, A-REX is usually started with the `a-rex` init script, which parses the `arc.conf` and creates an XML configuration, then starts the HED. This configuration uses the IdentityMap SecHandler to do an initial user mapping based on the configured grid-map file, if it fails, it maps to "nobody", then it uses the LegacySecHandler to match the user to Groups and VOs configured in arc.conf, then it uses the ArcAuthZ SecHandler with a LegacyPDP inside to allow or deny connections based on the authorization configured in the `[gridftpd/jobs]` section of the `arc.conf` (and the previously collected Group and VO information), then the LegacyMap SecHandler tries to map the Grid user to a local user based on the `[gridftpd]` section of `arc.conf` (and the previously collected Group and VO information).

6.13 Structure of the grid-mapfile

The following is not needed to setup a production environment but is described here as a reference.

A grid-mapfile is a simple text file. Each line is a record of the form

```
<grid identity certificate DN> <unix local account>
```

For each user that will connect to the CE, a Distinguished Name or DN contained in each user's certificate will be needed. Many Grid users can map to the same unix account.

A sample grid-mapfile is shown below:

```
"/DC=eu/DC=KnowARC/O=Lund University/CN=demo1" griduser1
"/DC=eu/DC=KnowARC/O=Lund University/CN=demo2" griduser1
"/DC=eu/DC=KnowARC/O=Lund University/CN=demo3" griduser2
"/DC=eu/DC=KnowARC/O=Lund University/CN=demo4" griduser2
"/DC=eu/DC=KnowARC/O=Lund University/CN=demo5" griduser2
```

Please refer to the certificate mini How-to to strip out the subject from Grid identity certificates.

6.14 Internal files of the A-REX

For each local UNIX user listed in the A-REX configuration – including a generic one which covers all local user identities – a *control directory* exists. In this directory the A-REX stores information about jobs belonging to that user. Multiple users can share the same *control directory*. In the most common configuration case, the A-REX serves all users defined by the Operating System and stores their control files in the same directory. To make it easier to recover in case of failure, the A-REX stores most information in files rather than in memory. All files belonging to the same job have names starting with **job.ID.,** where ID is the job identifier.

The files in the control directory and their formats are described below:

- *job.ID.status* – current state of the job. This is a plain text file containing a single word representing the internal name of current state of the job. Possible values and corresponding external job states are:
  - ACCEPTED
  - PREPARING

  - SUBMIT
  - INLRMS
  - FINISHING
  - FINISHED
  - CANCELING
  - DELETED

See Section 6.4 for a description of the various states. Additionally each value can be prepended the prefix "PENDING:" (like PENDING:ACCEPTED, see Section 6.4). This is used to show that a job is *ready* to be moved to the next state but it has to stay in it's current state *only* because otherwise some limits set in the configuration would be exceeded.

This file is not stored directly in the *control directory* but in the following sub-directories:

- accepting - for jobs in ACCEPTED state
- finished - for jobs in FINISHED and DELETED states
- processing - for other states
- restarting - temporary location for jobs being restarted on user request or after restart of A-REX

- *job.ID.description* – contains the description of the job (JD).
- *job.ID.local* – information about the job used by the A-REX. It consists of lines of format *"name = value"*. Not all of them are always available. The following names are defined:
  - *globalid* – job identifier as seen by user tools. Depending on used interface it is either BES ActivityIdentifier XML tree, GUID of EMI ES or GridFTP URL.
  - *headnode* – URL of service interface used to submit this job.
  - *interface* – name of interface used for jobs submission - *org.nordugrid.xbes*, *org.ogf.emies* or *org.nordugrid.gridftpjob*.
  - *lrms* – name of the LRMS backend to be used for local submission
  - *queue* – name of the queue to run the job at
  - *localid* – job id in LRMS (appears only after the job reached state **InLRMS**)
  - *args* – main executable name followed by a list of command-line arguments
  - *argscode* – code which main executable returns in case of success
  - *pre* – executable name followed by a list of command-line arguments for executable to run before main executable. There maybe few of them
  - *precode* – code which pre-executable returns in case of success
  - *post* – executable name followed by a list of command-line arguments for executable to run after main executable. There maybe few of them
  - *postcode* – code which post-executable returns in case of success
  - *subject* – user certificate's subject, also known as the distinguished name (DN)
  - *starttime* – GMT time when the job was accepted represented in the Generalized Time format of LDAP
  - *lifetime* – time period to preserve the SD after the job has finished in seconds
  - *notify* – email addresses and flags to send mail to about the job specified status changes
  - *processtime* – GMT time when to start processing the job in Generalized Time format
  - *exectime* – GMT time when to start job execution in Generalized Time format
  - *clientname* – name (as provided by the user interface) and IP address:port of the submitting client machine
  - *clientsoftware* – version of software used to submit the job

- *rerun* – number of retries left to rerun the job
- *priority* –
- *downloads* – number of files to download into the SD before execution
- *uploads* – number of files to upload from the SD after execution
- *rtes* –
- *jobname* – name of the job as supplied by the user
- *projectname* – name of the project as supplied by the user. There may be few of them
- *jobreport* – URL of a user requested *logger service*. The A-REX will also send job records to this service in addition to the default logger service configured in the configuration. There may be few of them
- *cleanuptime* – GMT time when the job should be removed from the cluster and it's SD deleted in Generalized Time format
- *expiretime* – GMT time when the credentials delegated to the job expire in Generalized Time format
- *gmlog* – directory name which holds files containing information about the job when accessed through GridFTP interface
- *sessiondir* – the job's SD
- *failedstate* – state in which job failed (available only if it is possible to restart the job)
- *failedcause* – contains *internal* for jobs failed because of processing error and *client* if client requested job cancelation.
- *credentialserver* – URL of MyProxy server to use for renewing credentials.
- *freestagein* – *yes* if client is allowed to stage-in any file
- *activityid* – Job-id of previous job in case the job has been resubmitted or migrated. This value can appear multiple times if a job has been resubmitted or migrate more than once.
- *migrateactivityid* –
- *forcemigration* – This boolean is only used for migration of jobs. It determines whether the job should persist if the termination of the previous job fails.
- *transfershare* – name of share used in **Preparing** and **Finishing** states.

This file is filled partially during job submission and fully when the job moves from the **Accepted** to the **Preparing** state.

- *job.ID.input* – list of input files. Each line contains 3 values separated by a space. First value contains name of the file relative to the SD. Second value is a URL or a file description. Example:

  input.dat gsiftp://grid.domain.org/dir/input_12378.dat

  A URL represents a location from which a file can be downloaded. Each URL can contain additional options.

  A file description refers to a file uploaded from the UI and consists of [size][.checksum] where

  size - size of the file in bytes.

  checksum - checksum of the file identical to the one produced by ***cksum*** (1).

  These values are used to verify the transfer of the uploaded file. Both size and checksum can be left out. A special kind of file description *.* is used to specify files which are **not** required to exist.

  The third optional value is path to delegated credentials to be used for communication with remote server.

  This file is used by the data staging subsystem of the A-REX. Files with *URL* will be downloaded to the SD or cache and files with 'file description' will simply be checked to exist. Each time a new **valid** file appears in the SD it is removed from the list and *job.ID.input* is updated.

- *job.ID.input_status* – contains list of files uploaded by client to the SD.

- *job.ID.output* – list of output files. Each line contains 1, 2 or 3 values separated by a space. First value is the name of the file relative to the SD. The second value, if present, is a URL. Supported URLs are the same as those supported by job.ID.input. Optional 3rd value is path to delegated credentials to be used while accessing remote server.

  This file is used by the data staging subsystem of the A-REX. Files with *URL* will be uploaded to SE and remaining files will be left in the SD. Each time a file is uploaded it is removed from the list and *job.ID.output* is updated. Files not mentioned as output files are removed from the SD at the the beginning of the **Finishing** state.

- *job.ID.output_status* – list of output files successfully pushed to remote locations.

- *job.ID.failed* – the existence of this file marks the failure of the job. It can also contain one or more lines of text describing the reason of failure. Failure includes the return code different from zero of the job itself.

- *job.ID.errors* – this file contains the output produced by external utilities like ***downloader***, ***uploader***, script for job submission to LRMS, etc on their stderr handle. Those are not necessarily errors, but can be just useful information about actions taken during the job processing. In case of problem include content of that file while asking for help.

- *job.ID.diag* – information about resources used during execution of job and other information suitable for diagnostics and statistics. It's format is similar to that of *job.ID.local*. The following names are at least defined:

  - *nodename* – name of computing node which was used to execute job,
  - *runtimeenvironments* – used runtime environments separated by ';',
  - *exitcode* – numerical exit code of job,
  - *frontend_distribution* – name and version of operating system distribution on frontend computer,
  - *frontend_system* – name of operating on frontend computer,
  - *frontend_subject* – subject (DN) of certificate representing frontend computer,
  - *frontend_ca* – subject (DN) of issuer of certificate representing frontend computer,

  and other information provided by GNU *time* utility. Note that some implementations of *time* insert unrequested information in their output. Hence some lines can have broken format.

- *job.ID.proxy* – delegated X509 credentials or chain of public certificates.

- *job.ID.proxy.tmp* – temporary X509 credentials with different UNIX ownership used by processes run with effective *user id* different from job owner's *id*.

- *delegations* – sub-directory containing collection of delegated credentials.

- *logs* – sub-directory with information prepared for reporting plugins.

There are other files with names like job.ID.* which are created and used by different parts of the A-REX. Their presence in the *control directory* can not be guaranteed and can change depending on changes in the A-REX code.

6.15 Environment variables set for the job submission scripts

The A-REX comes with support for several LRMS. Features explained below are for **PBS/Torque** backend, but for the other backends the behaviour is similar. This support is provided through *submit-pbs-job*, *cancel-pbs-job*, *scan-pbs-job* scripts. *submit-pbs-job* creates job's script and submits it to PBS. Created job's script is responsible for moving data between frontend machine and cluster node (if required) and execution of actual job. Alternatively it can download input files and upload output if *"localtransfer=no"* is specified in the configuration file.

Behavior of submission script is mostly controlled using environment variables. Most of them can be specified on frontend in A-REX's environment and overwritten on cluster's node through PBS configuration. Some of them may be set in configuration file too.

PBS_BIN_PATH – path to PBS executables. Like */usr/local/bin* for example. Corresponds to *pbs_bin_path* configuration command.

PBS_LOG_PATH – path to PBS server logs. Corresponds to *pbs_log_path* configuration command.

TMP_DIR – path to directory to store temporary files. Default value is */tmp*. Corresponds to *tmpdir* configuration command.

RUNTIME_CONFIG_DIR – path where runtime setup scripts can be found. Corresponds to *runtimedir* configuration command.

GNU_TIME – path to GNU time utility. It is important to provide path to utility compatible with GNU time. If such utility is not available, modify *submit-pbs-job* to either reset this variable or change usage of available utility. Corresponds to *gnu_time* configuration command.

NODENAME – command to obtain name of cluster's node. Default is */bin/hostname -f*. Corresponds to *nodename* configuration command.

RUNTIME_LOCAL_SCRATCH_DIR – if defined should contain path to the directory on computing node, which can be used to store job's files during execution. *scratchdir* configuration command.

RUNTIME_FRONTEND_SEES_NODE – if defined should contain path corresponding to *RUNTIME_LOCAL_SCRATCH_DIR* as seen on **frontend** machine. Corresponds to *shared_scratch* configuration command.

RUNTIME_NODE_SEES_FRONTEND – if set to "*no*" means computing node does not share file system with frontend. In that case content of the SD is moved to computing node by using means provided by the LRMS. Results are moved back after job's execution in a same way. Corresponds to *shared_filesystem* configuration command.

For the last options, see Section **6.16**, *Using a scratch area*

6.16 Using a scratch area

Figures 6.6, 6.7 and 6.8 present some possible combinations for RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_NODE and explain how data movement is performed. Figures a) correspond to the situation right after all input files are gathered in the session directory and actions taken right after the job script starts. Figures b) show how it looks while the job is running and actions which are taken right after it has finished. Figures c) show the final situation, when job files are ready to be uploaded to external storage elements or be downloaded by the user.

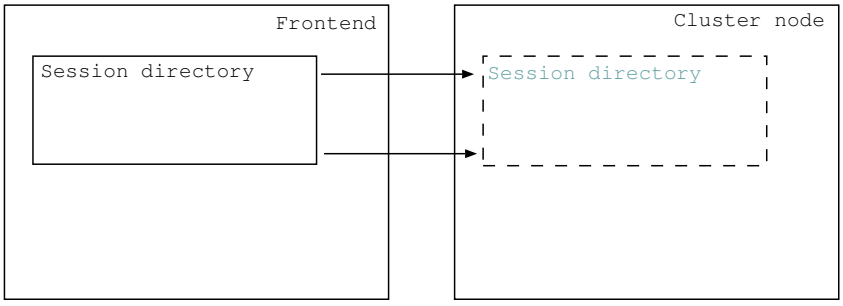


Figure 6.6: Both RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_NODE undefined. Job is executed in a session directory placed on the frontend.

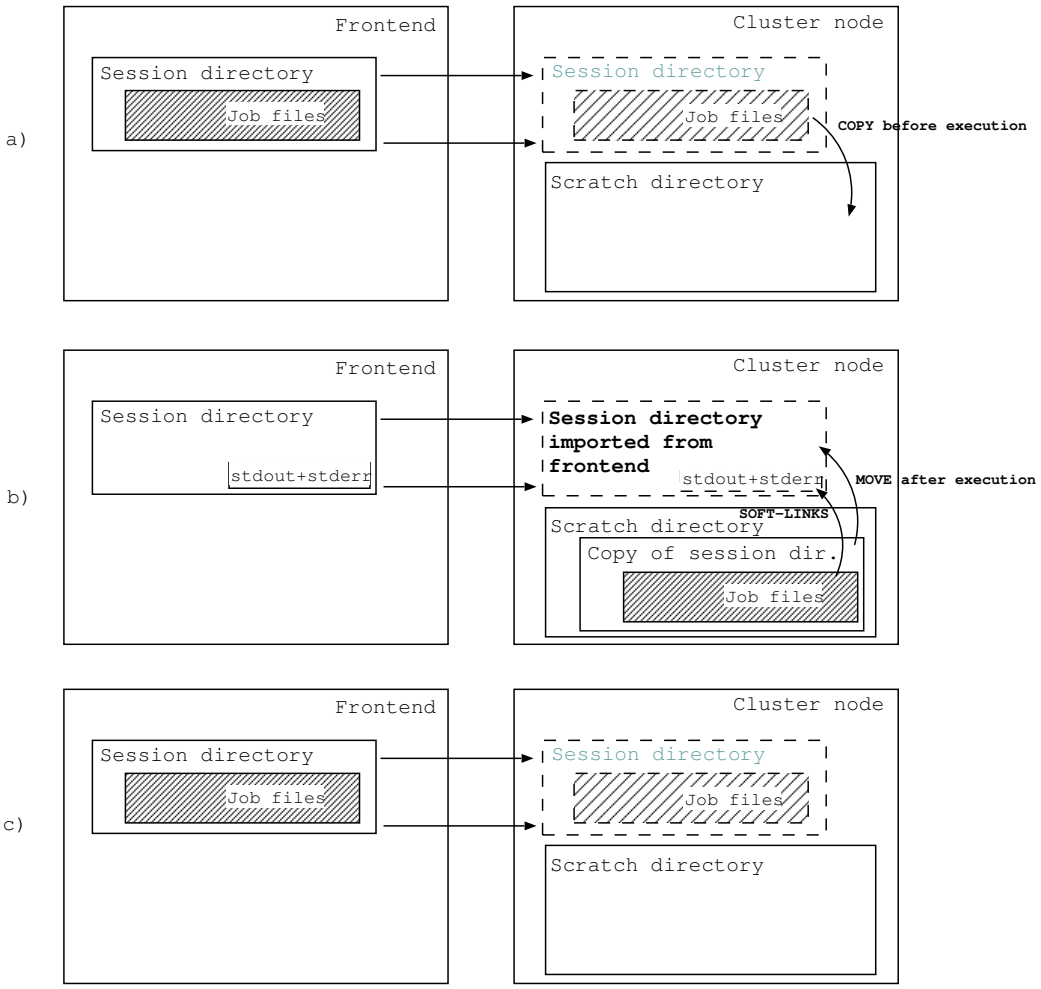


Figure 6.7: RUNTIME_LOCAL_SCRATCH_DIR is set to a value representing the scratch directory on the computing node, RUNTIME_FRONTEND_SEES_NODE is undefined.

a) After the job script starts all input files are moved to the "scratch directory" on the computing node.

b) The job runs in a separate directory in "scratch directory". Only files representing the job's *stdout* and *stderr* are placed in the original "session directory" and soft-linked in "scratch". After execution all files from "scratch" are moved back to the original "session directory".

c) All output files are in "session directory" and are ready to be uploaded/downloaded.

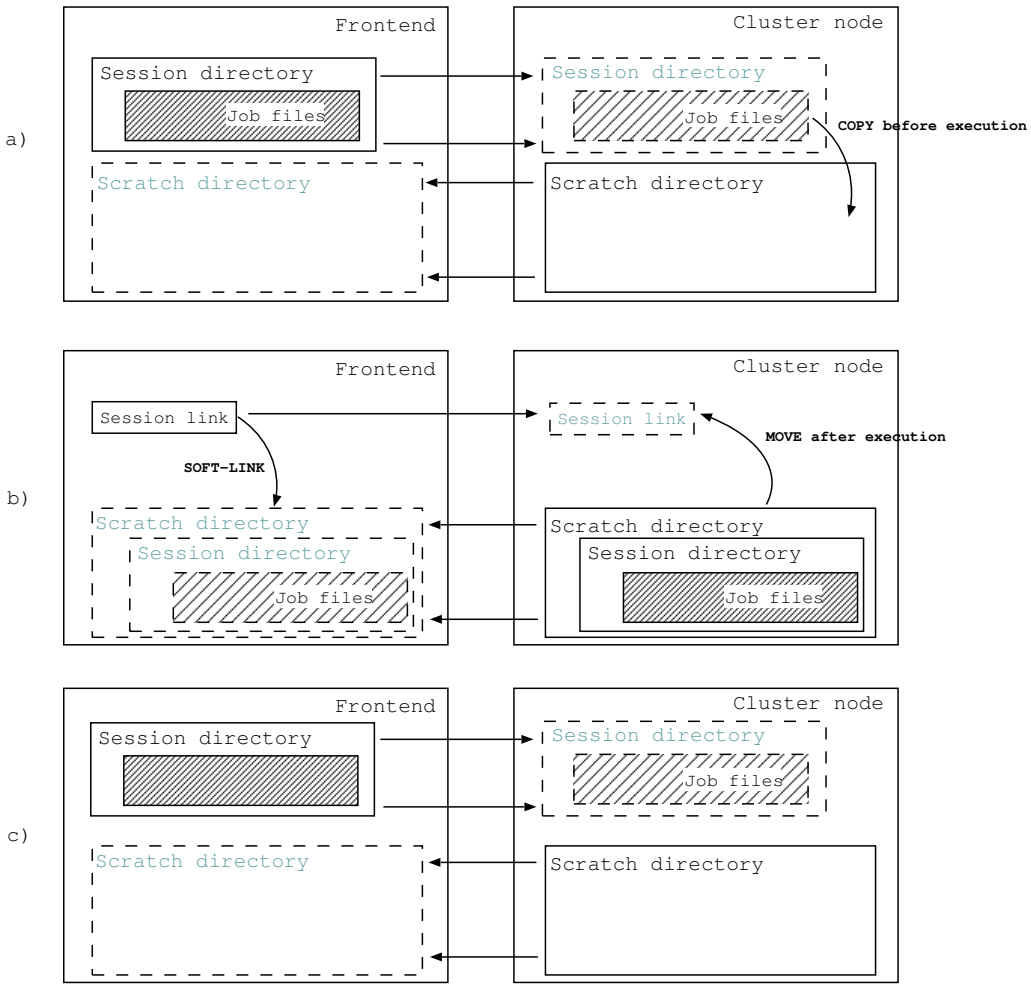


Figure 6.8: RUNTIME_LOCAL_SCRATCH_DIR and RUNTIME_FRONTEND_SEES_NODE are set to values representing the scratch directory on the computing node and a way to access that scratch directory from the frontend respectively.

a) After the job script starts, all input files are moved to "scratch directory" on the computing node. The original "session directory" is removed and replaced with a soft-link to a copy of the session directory in "scratch" as seen on the frontend.

b) The job runs in a separate directory in "scratch directory". All files are also available on the frontend through a soft-link. After execution, the soft-link is replaced with the directory and all files from "scratch" are moved back to the original "session directory"

c) All output files are in "session directory" and are ready to be uploaded/downloaded.

6.17 Web Service Interface

The A-REX Web Service Interface provides means to submit a description of a computational job to a computing resource, to stage-in additional data, to monitor and control processing of jobs, and obtain data produced during the execution of a job. The WS Interface is built and deployed inside the Hosting Environment Daemon (HED) infrastructure [**?**].

6.17.1 Basic Execution Service Interface

The job submission and control interface is based on a document produced by the OGF OGSA Basic Execution Services (BES) Working Group [**?**].

The exchange of SOAP messages is performed via HTTP(S). The BES interface is represented by two port-types – BES-Management and BES-Factory. The former is made to control the A-REX service itself and thus defines operations to start and stop the functionality of the BES service. The A-REX does not implement remote control of service functionality. Hence the BES-Management port-type is not functional. The BES-Factory port-type provides operations to submit new jobs (to create an activity in terms of BES) and to monitor its state. It also has an ability to provide information about the service. A-REX fully implements the functionality of this port-type.

For job descriptions A-REX accepts the Job Submission Description Language (JSDL) [**?**] documents as defined by the OGF Job Submission Description Language Working Group. Supported elements and extensions are described below.

6.17.2 Extensions to OGSA BES interface

A-REX introduces two new operations in addition to those provided by BES. It does that by defining its own port-type with new operations *ChangeActivityStatus* and *MigrateActivity*(see Appendix **??**).

The *ChangeActivityStatus* operation provides a way to request simple transfers between states of jobs and corresponding actions.

- *ChangeActivityStatus*
  - Input
    - *ActivityStatusType OldStatus*: Description of the state the job is supposed to be in during execution of this request. If the current state of the job is different from the one having been given, the operation is aborted and a fault is returned. This parameter is optional.
    - *ActivityStatusType NewStatus*: Description of the state the job is to be put into.
  - Output
    - *ActivityStatusType NewStatus*: Description of the current state of the job.
  - Fault(s)
    - *NotAuthorizedFault*: Indicates that the client is not allowed to do this operation.
    - *InvalidActivityIdentifierFault*: There is no such job/activity.
    - *CantApplyOperationToCurrentStateFault*: The requested transition is not possible.

  On result of this command, the job should be put into the requested state. If such a procedure cannot be performed immediately then the corresponding sequence is initiated and fault OperationWillBeAppliedEventuallyFault will be returned.

  Since BES allows implementations to extend their initial activity states with additional sub-states, A-REX defines a set of sub-states of activity processing in addition to those defined by the BES, as listed in Table 6.1. Their meaning is described in Section 6.4.

  The *MigrateActivity* operation generates a request to migrate a grid job from another A-REX, i.e. the operation will get input files and possibly job description from the cluster currently holding the job and create the job as a new activity at the present cluster. Currently only migration of queuing jobs is supported.

- *MigrateActivity*
  - Input
    - *wsa:EndpointReferenceType ActivityIdentifier*: This element should contain the *wsa:EndpointReference* of the job to be migrated.
    - *ActivityDocument*: JSDL document of the job to be migrated. This element is optional.
    - *Boolean ForceMigration*: Boolean that determines whether the job will persist on the new cluster if the termination of the previous job fails.
  - Output
    - *wsa:EndpointReferenceType ActivityIdentifier*: This element should contain the *wsa:EndpointReference* of the new activity.
    - *ActivityDocument*: Contains the JSDL document of the new activity.
  - Fault(s)
    - *NotAuthorizedFault*: Indicates that the client is not allowed to do this operation.
    - *NotAcceptingNewActivitiesFault*: A fault that indicates that A-REX currently is not accepting new activities.
    - *UnsupportedFeatureFault*: This fault indicates that an sub-element in the JDSL document is not supported or the ActivityDocument has not been recognised as JSDL.
    - *InvalidRequestMessageFault*: This fault indicates that an element in the request is either missing or has an invalid format. Typically this would mean that the job-id cannot be located in the ActivityIdentifier of the old job.

The *ActiviterIdentifier* specifies the URL of the job which will be migrated. In case the *ActivityDocument* is filled this document will be used to create a new activity otherwise an attempt will be made to retrieve the job description through the BES operation *GetActivityDocument*.

Once the input files have been downloaded from the other cluster, a request will be send to terminate the old job. If this request fails the new activity at the present cluster will be terminate unless the *ForceMigration* is true. This is to prevent the job from being executed at two different places at the same time.

6.17.3 Delegation Interface

The A-REX also supports the Delegation Interface (see Appendix **??**). This is a common purpose interface to be used by ARC services which accepts delegated credentials from clients. The Delegation Interface implements two operations: initialization of credentials delegation (DelegateCredentialsInit) and update/renewal of credentials (UpdateCredentials).

- *DelegateCredentialsInit* operation – this operation performs the first half of the credentials delegation sequence.
  - Input
    - None. On this request the service generates a pair of *public* and private keys. The public key is then sent to the client in response.
  - Output(s)
    - *TokenRequestType TokenRequest*: Contains the public key generated by the service as a Value element. It also provides an identifier in the Id element which should be used to refer to the corresponding private key.
  - Fault(s)
    - *UnsupportedFault*: Indicates that the service does not support this operation despite supporting the port-type.
    - *ProcessingFault*: Internal problems during generation of the token.
- *UpdateCredentials* operation – this operation makes it possible to update the content of delegated credentials (like in the case of credentials being renewed) unrelated to other operations of the service.

Table 6.1: Job states definitions and mappings

Applicable BES state	ARC BES sub-state	EMI ES state	ARIS state	A-REX internal state	Description
Pending	Accepting	ACCEPTED		ACCEPTED	Job is in the process submitted. This stat recognised by the A-R *Accepted* is first reporte
	Accepted	ACCEPTED		ACCEPTED	Job was submitted
Running	Preparing	PREPRO-CESSING		PREPARING	Stage-in process is goin
	Prepared	PREPRO-CESSING		PREPARING + PENDING	Stage-in process has fin
	Submitting	PROCESSING-ACCEPTING		SUBMIT	Communication with lo system is in process
	Queued	PROCESSING-RUNNING		INLRMS	Job entered local batc but is not runnning nc state is not recognised l REX yet. *Executing* is instead
	Executing	PROCESSING-RUNNING		INLRMS	Job is being executed batch system
	Executed	PROCESSING-RUNNING		INLRMS, INLRMS + PENDING	Job execution in local b tem has finished. The dos not detect job stat local batch system yet sult this state is reporte job is *Pending*.
	Killing	PROCESSING		CANCELING	Communication with lo system to terminate exe in process
	Finishing	POSTPRO-CESSING		FINISHING	Stage-out process is goi
Cancelled	Killed	TERMINAL		FINISHED	Job was stopped by exp request. The A-REX does not remember this *Failed* is reported inste
Failed	Failed	TERMINAL		FINISHED	There was a failure du cution
Finished	Finished	TERMINAL		FINISHED	Job finished successfull
Finished	Deleted	TERMINAL		DELETED	Job finished and was l REX too long
All	Pending			PENDING	Job is prevented from the next state due to so nal limits; this sub-stat in parallel with other s
All	Held				Job processing is susp client request; this sub- pears in parallel with o states. This state is res future and is not imp yet.

- Input
  - *DelegatedTokenType DelegatedToken*: Contains an X509 proxy certificate based on the public key from the DelegateCredentialsInit signed by the user's proxy certificate. Also includes the Id element which identifies the private key stored at the service side associated with these credentials. The reference element refers to the object to which these credentials should be applied in a way specific to the service. The same element must also be used for delegating credentials as part of other operations on service.
- Output(s)
  - None.
- Fault(s)
  - *UnsupportedFault*: Indicates that service does not support this operation despite supporting the port-type.
  - *ProcessingFault*: Internal problems during generation of the token.

Additionally, A-REX Web Service Interface allows delegation to be performed as part of the *CreateActivity* operation of the BES-Factory port-type. For this it accepts the element *DelegatedCredentials* inside the *CreateActivity* element. The *Id* element of *DelegatedCredentials* must contain an identifier obtained in response to the previous *DelegateCredentialsInit* operation. For more information about delegations and delegation interface refer to [**?**].

6.17.4 Local Information Description Interface

The A-REX implements the Local Information Description Interface (LIDI) interface common for all ARC services. This interface is based on OASIS Web Services Resource Properties specification [**?**]. Information about resources and maintained activities/jobs are represented in a *WS-Resource Properties* informational XML document. The document type is defined in the A-REX WSDL as a *ResourceInformationDocument-Type*. It contains the following elements/resources:

nordugrid – description of computing resource that uses NorudGrid LDAP schema [**?**] converted to XML document.

Domains – description of a computation resource that uses Glue2 schema.

All information can be accessed either through requests on particular resources or through XPath queries using WS-Resource Properties operations.

6.17.5 Supported JSDL elements

A-REX supports the following elements from the JSDL version 1.0 specification [**?**] including POSIX Applications extension and JSDL HPC Profile Application Extension [**?**]:

JobName – name of the job as assigned by the user.

Executable (POSIX,HPC) – name of the executable file.

Argument (POSIX,HPC) – arguments the executable will be launched with.

DataStaging

Filename – name of the data file on the executing node.

Source – source where the file will be taken from before execution.

Target – destination the file will be delivered to after execution.

Input (POSIX,HPC) – file to be used as standard input for the executable.

Output (POSIX,HPC) – file to be used as standard output for the executable.

Error (POSIX,HPC) – file to be used as standard error for the executable.

MemoryLimit (POSIX) – amount of physical memory needed for execution.

TotalPhysicalMemory – same as MemoryLimit.

IndividualPhysicalMemory – same as MemoryLimit.

CPUTimeLimit (POSIX) – maximal amount of CPU time needed for execution.

TotalCPUTime – same as CPUTimeLimit.

IndividualCPUTime – same as CPUTimeLimit.

WallTimeLimit (POSIX) – amount of clock time needed for execution.

TotalCPUCount – number of CPUs needed for execution.

IndividualCPUCount – same as *TotalCPUCount*.

6.17.6 ARC-specific JSDL Extensions

A-REX accepts JSDL documents having the following additional elements (see Appendix **??**):

IsExecutable – marks file to become executable after being delivered to the computing resource.

RunTimeEnvironment – specifies the name of the Runtime Environment needed for job execution.

Middleware – request for specific middleware on the computing resource frontend.

RemoteLogging – destination for the usage record report of the executed job.

LocalLogging – name for the virtual directory available through job interface and containing various debug information about job execution.

AccessControl – ACL expression which describes the identities of those clients who are allowed to perform operations on this job.

Notify – Email destination for notification of job state changes.

SessionLifeTime – duration for the directory containing job-related files to exist after the job finished executing.

JoinOutputs – specifies if standard output and standard error channels must be merged.

Reruns – defines how many times a job is allowed to rerun in case of failure.

CredentialServer – URL of MyProxy service which may be used for renewing the expired delegated job credentials.

CandidateTarget – specifies host name and queue of a computing resource.

OldJobID – specifies the previous job-ids in case the job has been resubmitted or migrated.

Acknowledgements

This work was supported in parts by: the Nordunet 2 program, the Nordic DataGrid Facility, the EU KnowARC project (Contract nr. 032691), the EU EMI project (Grant agreement nr. 261611) and the Swedish Research council via the eSSENCE strategic research program.

Bibliography