NorduGrid ARC 6 Information

Release ARC6

NorduGrid Collaboration

Jul 12, 2019
The *Advanced Resource Connector (ARC)* middleware, developed by the NorduGrid Collaboration, is an open source software solution enabling e-Science computing infrastructures with emphasis on processing of large data volumes. ARC is being used to enable national and international e-infrastructures since its first release in 2002.

This document is dedicated to the ARC Version 6 collecting all relevant information in one place. You should be able to find information regarding the code, documentation, testing activities, support channels, ... and so on here. The information is refreshed daily, a snapshot of the development version can be found here.

If you are new to ARC start reading the *Try ARC6* quickstart guide to get an overview of main operations in the simple test case.

For production *Computing Element* deployment follow the *Installation and Configuration Guide* that contains the structure and pointers to precise configuration of every ARC subsystem.

In case you are migrating to ARC 6 from an ARC 5 installation read the *Migration Guide*. For an overview of the main changes compared to ARC 5, please visit *Main changes in ARC 6 compared to ARC 5*.

The ultimate description of the new ARC 6 configuration can be found in the *ARC Configuration Reference Document*. 
ARC is available for variety of GNU/Linux flavors via stable Repositories or Nightly Builds if you want to test the latest development release.

The source code is hosted in NeIC’s Coderefinery GitLab repository.
User support and site installation assistance is provided via the nordugrid-discuss mailing list, and the Nordugrid Bugzilla.
This section contains a documentation about all ARC middleware services deployment, configuration and operations. If you are looking for ARC Computing Element setup instruction or performance tuning parameters you are in the right place.

3.1 ARC Configuration Reference Document

3.1.1 General configuration structure

This is the arc.conf REFERENCE DOCUMENT defining the configuration blocks and configuration options for the ARC services.

The arc.conf configuration file consists of the following blocks:

[common]
[authgroup:groupname]
[mapping]
[lrms]
[lrms/ssh]
[arex]
[arex/cache]
[arex/cache/cleaner]
[arex/data-staging]
[arex/ws]
[arex/ws/jobs]
[arex/ws/cache]
[arex/ws/candypond]
[arex/ws/argus]
[arex/jura]
[arex/jura/archiving]
[arex/jura/sgas:targetname]
[arex/jura/ape1:targetname]
[arex/ganglia]
[gridftp]
[gridftp/jobs]
[gridftp/filedir]
[infosys]
[infosys/ldap]
[infosys/nordugrid]
[infosys/glue2]
[infosys/glue2/ldap]
[infosys/glue1]
[infosys/glue1/site-bdii]
[infosys/cluster]
[queue:name]
[datadelivery-service]
[acix-scanner]

(continues on next page)
A block configures an ARC service, a service interface, a utility or a subsystem. Enabling (turning on) a functionality, a service or an interface requires the presence of the appropriate configuration block. To disable a service or an interface, simply delete or comment out the related `arc.conf` block (you may need to rerun the corresponding startup script).

The `[common]` block is mandatory even if not a single option is specified within. The presence of the block turns on the default values for the configuration options within the block.

As an example, in order to set up a minimalistic ARC CE offering no external interfaces you need to configure at least the `[common]`, `[mapping]`, `[arex]`, `[lrms]`, `[infosys]` and `[queue:name]` blocks.

As another example, an ARC-based data offloader would require the `[common]` and the `[datadelivery-service]` blocks.

A block is identified by its block header. A block header may consist of keywords and optionally block identifiers. Keywords may be separated by `/` and used to label subblocks (e.g. `[arex/jura]`), while block identifiers are separated by `:` from keywords. For example, in the `[queue:short]` block header `queue` is a keyword while `short` is an identifier, e.g. the name of the queue. Block headers must be UNIQUE.

A block starts with a unique `[keyword:identifier]` blockheader and ends where the next block starts, that is at the next `[blockheader]` directive.

A block may have sub-blocks e.g. the various interfaces of the AREX service are configured via sub-blocks (e.g. `[arex/ws]`). When a sub-block is enabled then the corresponding parent block must also appear in the `arc.conf` file.

Configuration blocks contain (config option, config value) pairs following the syntax in single line:

```plaintext
config_option=value element [optional value element]
```

**Note:** quotes around the configuration value(s) must NOT be used any longer.

**Note:** the `arc.conf` is CASE-SENSITIVE!

Space handling syntax in `arc.conf` for configuration lines:

```plaintext
(stripped space)option(stripped space)=(stripped space)value(saved_value)
```

and for block headers:

```plaintext
[keyword:(stripped space)space is NOT allowed within identifier(stripped space)]
```

Detailed textual definition:

1. All trailing and leading spaces on each configuration line are stripped and ignored. This applies both to block headers and block content.

2. All spaces around the `=` sign in `option=value` kind of string (after ‘a’ is applied) are stripped and ignored. For example line `hostname = myhost.info` is treated as identical to `hostname=myhost.info`. 
3. In block headers of [keyword] kind (after ‘a’ is applied) no additional spaces are allowed around keyword and inside keyword.

4. In block headers of [keyword:identifier] kind (after ‘a’ is applied) no additional spaces are allowed around keyword and inside both keyword and identifier. The spaces ARE allowed around identifier part and stripped and ignored.

Mandatory configuration options are indicated by an asterix prefix to the option name e.g: *mandatory_configoption. Mandatory options with undefined values will result in service stop during the startup process.

Each of the configuration options have well-defined default that is specified in this reference file. The default can take either a pre-set value, a special substitution or the keyword undefined. Configuration options within an enabled block take their default values in case they are missing (or commented out). Configuration parameters with undefined defaults takes no values. Furthermore, configuration options within disabled blocks takes no values either.

Configuration blocks related to authorization are ORDER-DEPENDENT! The authorization blocks [authgroup:name] MUST be defined before used in the blocks such as [mapping],[arex/ws/jobs] or [gridftp/jobs]. The same rule applies to defining legacy [userlist:name] blocks. Furthermore, the order of the authorization blocks itself may have influence over authorization decisions!

Below we give a detailed description of all the configuration options of the different configuration blocks. Every configuration option is described in a dedicated paragraph with the following reference syntax notation. This file is parsed at buildtime to assist in configuration default parsing and validation script and so it is important that it follows the agreed syntax: For each block or option please add explanatory text with two ## followed by a space at the beginning of the line and then an example with a single # and no spaces at the beginning of the line.

**example_config_option**

**Synopsis:** example_config_option = value [optional values]

**Description:** Here comes the explanation of the config option. Mandatory configuration options are indicated by an asterix prefix to the option name e.g: *mandatory_configoption vs. optional_configoption.

The explanation can be followed by the special keywords in a separate line:

- **multivalued** - used to indicate that config option can be specified multiple times. This forms a set of values for the same configuration option irrespective of lines order.

- **sequenced** - used to indicate that config option is a part of the sequence and its effect on configuration depends on the lines order. Sequenced option can be specified several times in the configuration sequence independently.

Missing such keywords means the config option can only occur once in the arc.conf. By default the arc.conf config options are optional and single-valued. For some config options only a fix set of values are allowed. These are listed in a separate line after the allowedvalues keyword. The default of every config option is explicitly given in the default: line. Default can be a pre-set value, a substitution or the undefined keyword. The last line of the paragraph is always a valid example preceded by a single #

This option in multivalued.

**Allowed values:** 12, 34, 56

**Default:** 34

**Example:**

```
example_config_option=56
```

### 3.1.2 Configuration blocks and options
[common] block

Common configuration affecting all ARC components, usually related to networking or security or service behaviour. This block is mandatory. The common block options may be overridden by the specific sections of the components later. The [common] always appears at the beginning of the config file. The config options set within this block are available for all the other blocks thus shared by the different components of ARC.

hostname

Synopsis: hostname = string
Description: The FQDN of the frontend on which the ARC services are deployed.
Default: $EXEC{hostname -f}
Example:

```
hostname=myhost.org
```

Warning: CHANGE: modified semantics, not mandatory any longer

http_proxy

Synopsis: http_proxy = url
Description: The http proxy server. This setting affects all client HTTP(s) requests that initiated by ARC core services, including data staging, SAML communications, and pushing SGAS accounting records. This variable is similar to setting the ARC_HTTP_PROXY environmental variable.
Default: undefined
Example:

```
http_proxy=proxy.mydomain.org:3128
```

Warning: CHANGE: moved here from the old [grid-manager] block

x509_host_key

Synopsis: x509_host_key = path
Description: (previously x509_user_key) Server credential location. Sets the full path to the host private key. These variables are similar to the GSI environment variable X509_USER_KEY. If indicated, the variable can be set individually for each service/component in the corresponding block.
Default: /etc/grid-security/hostkey.pem
Example:

```
x509_host_key=/etc/grid-security/hostkey.pem
```

Warning: CHANGE: renamed
x509_host_cert

Synopsis: x509_host_cert = path

Description: (previously x509_user_cert) Server credential location. Sets the full path to the host public certificate. These variables are similar to the GSI environment variable X509_USER_CERT. If indicated, the variable can be set individually for each service/component in the corresponding block.

Default: /etc/grid-security/hostcert.pem

Example:

x509_host_cert=/etc/grid-security/hostcert.pem

Warning: CHANGE: renamed

x509_cert_dir

Synopsis: x509_cert_dir = path

Description: Location of trusted CA certificates. This variable is similar to the GSI environment variable X509_CERT_DIR. If indicated, the variable can be set individually for each service/component in the corresponding block.

Default: /etc/grid-security/certificates

Example:

x509_cert_dir=/etc/grid-security/certificates

x509_voms_dir

Synopsis: x509_voms_dir = path

Description: The path to the directory containing *.lsc files needed for verification of VOMS service signature in the proxy-certificate.

Default: /etc/grid-security/vomsdir

Example:

x509_voms_dir=/etc/grid-security/vomsdir

voms_processing

Synopsis: voms_processing = keyword

Description: Defines how to behave if errors in VOMS AC processing detected. The following keywords are supported:

- **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.
Allowed values: relaxed, standard, strict, noerrors
Default: standard
Example:

voms_processing=strict

[authgroup:groupname] block

These configuration blocks contain authorization rules. An [authgroup:groupname] block always defines a group of users where members of the group are those who satisfy the authorization rules. The rules within the block determine which user belong to the authgroup. Then, access control and identity mapping of ARC services are implemented via associating a authgroup with an interface, queue or a mapping rule using one of the allowaccess, denyaccess or [mapping] block parameters. For more info please read Security Framework of ARC at http://www.nordugrid.org/documents/arc-security-documentation.pdf The authgroup should not be mistaken for a virtual organisation (VO). An authgroup may match a single VO if only a single check (rule) on VO membership is perfomed.

IMPORTANT: Rules in an authgroup are processed in their order of appearance. The first matching rule decides the membership of the user to the authgroup being evaluated and the processing STOPS within that authgroup. This does not mean that the same user is not processed for the next authgroup: all [authgroup:groupname] blocks are evaluated, even if a user already has a match with one of the earlier groups.

All the objects used in the rules MUST be defined before it may be used. For example, to create group of authgroups you must first defined the child groups.

There are positively and negatively matching rules. If a rule is matched positively then the user tested is accepted into the respective group and further processing is stopped. Upon a negative match the user would be rejected for that group - processing stops too. The sign of rule is determined by prepending the rule with + (for positive) or - (for negative) signs. + is default and can be omitted. A rule may also be prepended with ! to invert result of rule, which will let the rule match the complement of users. That complement operator (!) may be combined with the operator for positive or negative matching.

Warning: CHANGE: renamed the block.

subject

Synopsis: subject = certificate_subject
Description: Rule to match specific subject of user’s X.509 certificate. No masks, patterns and regular expressions are allowed.
This is sequenced option.
Default: undefined
Example:

subject=/O=Grid/O=Big VO/CN=Main Boss
subject=/O=Grid/O=Big VO/CN=Deputy Boss

file

Synopsis: file = path
Description: Processes a list of DNs stored in an external file one per line in grid-mapfile format (see map_with_file from [mapping] block, unixname is ignored) and adds those to the authgroup.
This is **sequenced** option.

**Default:** undefined

**Example:**

```
file=/etc/grid-security/local_users
file=/etc/grid-security/atlas_users
```

**Warning:** CHANGE: Modified semantics. The external file should only contain DNs, no complex rules. No need to change the code.

### voms

**Synopsis:** voms = vo group role capabilities

**Description:** Match VOMS attribute in user’s credential. Use * to match any value.

This is **sequenced** option.

**Default:** undefined

**Example:**

```
voms=nordugrid Guests * *
voms=atlas students prodman *
```

### authgroup

**Synopsis:** authgroup = group_name [group_name ...]

**Description:** (previously group) Match user already belonging to one of specified authgroups. The authgroup referred here must be defined earlier in arc.conf configuration file. Multiple authgroup names may be specified for this rule. That allows creating hierarchical structure of authorization groups like all-atlas are those which are atlas-users and atlas-admins.

This is **sequenced** option.

**Default:** undefined

**Example:**

```
authgroup=local_admins
authgroup=local_admins remote_users
```

**Warning:** CHANGE: renamed

### userlist

**Synopsis:** userlist = ulist_name [ulist_name ...]

**Description:** (previously vo) Match user belonging to ulist_name defined in an earlier [userlist:ulist_name] block. Multiple userlist names are allowed for this rule.

This is **sequenced** option.

**Default:** undefined

**Example:**
userlist=biousers

**Warning:** CHANGE: renamed

**plugin**

*Synopsis:* plugin = timeout path [arg1 [arg2 [arg3...]]]

*Description:* Run external executable or function from shared library. Rule is matched if plugin returns 0. In arguments following substitutions are supported:

- `%D` - subject of certificate
- `%P` - path to proxy

ARC ships with LCAS plugin that can be enabled with following plugin configuration. For more information about configuring LCAS refer to `Using LCAS/LCMAPS` document.

This is **sequenced** option.

*Default:* undefined

*Example:*

```
plugin=10 /usr/libexec/arc/arc-lcas %D %P liblcas.so /usr/lib64 /etc/lcas/lcas.db
```

**all**

*Synopsis:* all = yes|no

*Description:* Matches any or none user identity. For *yes* argument this rule always returns positive match. For *no* it is always no match.

This is **sequenced** option.

*Default:* undefined

*Example:*

```
all=yes
```

**Warning:** CHANGE: MODIFIED options

**[mapping] block**

This block defines the grid-identity to local UNIX identity mapping rules used by various ARC components.

Rules in the `[mapping]` block are processed IN A SEQUENCE in line order of the configuration file (from top to bottom).

There are two kind of rules:

- mapping rules that defines how the particular authgroup members are mapped
- policy rules that modifies the mapping rules sequence processing

Default policy for mapping rules processing is:

- processing CONTINUES to the next rule if identity of user DO NOT match authgroup specified in the rule (can be redefined with `policy_on_nogroup` option)
• processing STOPS if identity of user match authgroup specified in the mapping rule. Depend on whether this mapping rule returns valid UNIX identity the processing can be redefined with policy_on_map and policy_on_nomap options.

Policy can be redefined at the any point of configuration sequence and affects all mapping rules defined after the policy rule.

**Note:** if mapping process STOPS and there is still no local UNIX identity identified, the user running A-REX will be used.

**Note:** when grid-identity is mapped to root account - request processing fails implicitly.

**Warning:** CHANGE: This is a new block

### map_to_user

**Synopsis:** map_to_user = authgroup_name unixname[:unixgroup]

**Description:** the users that belongs to specified authgroup are mapped to unixname local UNIX account that may be optionally followed by a unixgroup UNIX group. In case of non-existing unixname account the mapping rule treated as a rule that did not returned mapped identity (nomap).

This is **sequenced** option.

**Default:** undefined

**Example:**

```
map_to_user=any nobody:nobody
```

### map_to_pool

**Synopsis:** map_to_pool = authgroup_name directory

**Description:** the user that belong to specified authgroup is assigned one of the local UNIX accounts in the pool. Account names that are part of this pool are stored line-by-line in the pool file inside the directory. The directory also contains information about used account names stored in another files. If there are no more available accounts in the defined pool for mapping then accounts not used for a configurable time period may be reassigned. The pool behaviour, including account reuse, is configureable with the optional directory/config file that has INI syntax (line-by-line key=value). Possible keys of the config file are:

- **timeout** Define the timeout in days (default is 10) after which the UNIX account can be reassigned to another user if not used. The 0 value means no lease expiration.

This is **sequenced** option.

**Default:** undefined

**Example:**

```
map_to_pool=atlas /etc/grid-security/pool/atlas
```

### map_with_file

**Synopsis:** map_with_file = authgroup_name file
**Description:** for users that belongs to specified authgroup the DN of certificate is matched against a list of DNs stored in the specified file, one per line followed by a local UNIX account name. The DN must be quoted if it contains blank spaces. This rule can be used to implement legacy grid-mapfile approach.

This is **sequenced** option.

**Default:** undefined

**Example:**

```
map_with_file=any /etc/grid-security/grid-mapfile
```

### map_with_plugin

**Synopsis:** map_with_plugin = authgroup_name timeout plugin [arg1 [arg2 [...]]]

**Description:** run external plugin executable with specified arguments to find the UNIX account name to which users that belongs to specified authgroup will be mapped to. A rule matches if the exit code is 0 and there is a UNIX account name printed on stdout (optionally followed by a UNIX group name separated by colon). Plugin execution time is limited to timeout seconds.

In the arguments the following substitutions are applied before the plugin is started:

- `%D` - subject of user’s certificate,
- `%P` - path to credentials’ proxy file.

ARC ships with LCMAPS plugin that can be enabled with the corresponding configuration. For more information about configuring LCMAPS refer to ‘Using LCAS/LCMAPS’ document.

This is **sequenced** option.

**Default:** undefined

**Example:**

```
map_with_plugin=altas 30 /usr/libexec/arc/arc-lcmaps %D %P liblcmaps.so /usr/lib64 → /etc/lcmaps/lcmaps.db arc
```

### policy_on_nomap

**Synopsis:** policy_on_nomap = continue/stop

**Description:** redefines mapping rules sequence processing policy in case identity of user match authgroup specified in the mapping rule and mapping rule DO NOT return valid UNIX identity. Default policy is stop processing the further rules. For example this policy will be triggered if pool is depleted, certificate subject is missing in the map file used for defined authgroup or plugin execution failed.

This is **sequenced** option.

**Default:** undefined

**Allowed values:** continue, stop

**Example:**

```
policy_on_nomap=continue
```

### policy_on_map

**Synopsis:** policy_on_map = continue/stop
**Description:** redefines mapping rules sequence processing policy in case identity of user match authgroup specified in the mapping rule and mapping rule return valid UNIX identity. Default policy is stop processing the further rules. This policy will be triggered if rule successfully returns the result (allocated in pool, matched in map file, plugin call was successful).

This is **sequenced** option.

**Default:** undefined

**Allowed values:** continue, stop

**Example:**

```plaintext
policy_on_map=stop
```

### policy_on_nogroup

**Synopsis:** policy_on_nogroup = continue/stop

**Description:** redefines mapping rules sequence processing policy in case identity of user DO NOT match authgroup specified in the mapping rule. Default policy is continue processing the further rules.

This is **sequenced** option.

**Default:** undefined

**Allowed values:** continue, stop

**Example:**

```plaintext
policy_on_nogroup=stop
```

### [lrms] block

This block specifies the characteristics of the Local Resource Manager System (batch system) underneath the ARC CE. This block contains all the lrms-specific parameters and information. Configuration values in this block are available for A-REX, the backends, accounting and infosys ARC subsystems.

ARC support the most common LRMS flavours.

**Warning:** CHANGE: This is a new block. Contains parameters previously set in [common], [infosys], [cluster], [queue]

### lrms

**Synopsis:** *lrms = lrmstype [defaultqueue]*

**Description:** Sets the type of the LRMS (queue system) and optionally the default queue name. ONLY ONE LRMS IS ALLOWED. MULTIPLE LRMS ENTRIES WILL TRIGGER UNEXPECTED BEHAVIOUR.

For lrmstype, the following values can be chosen:

- fork - simple forking of jobs to the same node as the server
- sge - (Sun/Oracle) Grid Engine
- condor - Condor
- pbs - PBS (covers Torque and other old PBS flavours e.g. OpenPBS, older PBSPro, etc)
- pbspro - Altair PBS Professional
- lsf - LSF
• ll - LoadLeveler
• slurm - SLURM
• boinc - Boinc
• slurmpy - new EXPERIMENTAL SLURM scripts (contains the ssh remote batch management as well)

The optional \texttt{defaultqueue} parameter specifies the name of an existing LRMS queue in the cluster that will be used by AREX as the default queue to submit grid jobs in case the client does not specify a queue name during the job submission process. This queue name must match one of the \texttt{[queue:queue\_name]} blocks.

\textit{Allowed values:} fork, sge, condor, pbs, pbspro, lsf, ll, slurm, boinc, slurmpy

\textit{Default:} undefined

\textbf{Example:}

\begin{verbatim}
lrms-pbspro gridlong
lrms-slurm
\end{verbatim}

\textbf{Warning:} CHANGE: new \texttt{lrmtystype} values added (slurmpy).

\textbf{lrmsconfig}

\textit{Synopsis:} \texttt{lrmsconfig = text}

\textit{Description:} An optional free text field to describe the configuration of your Local Resource Management System (batch system). The value is published in the infosys, and is not used otherwise.

\textit{Default:} undefined

\textbf{Example:}

\begin{verbatim}
lrmsconfig=single job per processor
\end{verbatim}

\textbf{Warning:} CHANGE: moved here from the \texttt{[cluster]} block

\textbf{defaultmemory}

\textit{Synopsis:} \texttt{defaultmemory = number}

\textit{Description:} The LRMS memory request of job to be set by the LRMS backend scripts, if a user submits a job without specifying how much memory should be used. The order of precedence is: job description -> default-memory. This is the amount of memory (specified in MB) that a job will request.

\textit{Default:} undefined

\textbf{Example:}

\begin{verbatim}
defaultmemory=512
\end{verbatim}

\textbf{Warning:} CHANGE: moved here from the \texttt{[cluster]} block
nodename

Synopsis: nodename = path

Description: Redefine the command to obtain hostname of LRMS worker node. By default the value is defined on buildtime and depend on the OS. In most cases /bin/hostname -f will be used.

Note: this way of getting WN hostname will be used only in case of particular LRMS backend had no native LRMS-defined way.

Default: undefined
Example:

```bash
nodename=/bin/hostname -s
```

Warning: CHANGE: moved here from the [arex] block

gnu_time

Synopsis: gnu_time = path

Description: Path to the GNU time command on the LRMS worker nodes. If time command exists on the node, jobscript will write additional diagnostic information.

Default: /usr/bin/time
Example:

```bash
gnu_time=/usr/bin/time
```

Warning: CHANGE: moved here from the [arex] block

pbs_bin_path

Synopsis: pbs_bin_path = path

Description: The path to the qstat,pbsnodes,qmgr etc PBS binaries, no need to set if PBS is not used.

Default: /usr/bin
Example:

```bash
pbs_bin_path=/usr/bin
```

pbs_log_path

Synopsis: pbs_log_path = path

Description: 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 qstat for that.

Default: /var/spool/pbs/server_logs
Example:
pbs_log_path=/var/spool/pbs/server_logs

**pbs_dedicated_node_string**

**Synopsis:** `pbs_dedicated_node_string = string`

**Description:** (previously dedicated_node_string) 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 available for grid jobs, and these nodes have a common node property string, this case the 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. You don’t need to set this attribute if your cluster is fully available for the grid and your cluster’s PBS config does not use the node property method to assign certain nodes to grid queues. You shouldn’t use this config option unless you make sure your PBS config makes use of the above described setup.

**Default:** undefined

**Example:**

```
pbs_dedicated_node_string=gridnode
```

**Warning:** CHANGE: renamed. moved here from [cluster]

**condor_bin_path**

**Synopsis:** `condor_bin_path = path`

**Description:** Path to Condor binaries. Must be set if Condor is used.

**Default:** /usr/bin

**Example:**

```
condor_bin_path=/opt/condor/bin
```

**condor_config**

**Synopsis:** `condor_config = path`

**Description:** Full path to Condor config file. Must be set if Condor is used and the config file is not in its default location (/etc/condor/condor_config or ~/condor/condor_config). The full path to the file should be given.

**Default:** /etc/condor/condor_config

**Example:**

```
condor_config=/opt/condor/etc/condor_config
```

**condor_rank**

**Synopsis:** `condor_rank = ClassAd_float_expression`

**Description:** If you are not happy with the way Condor picks nodes when running jobs, you can define your own ranking algorithm by optionally setting the condor_rank attribute. condor_rank should be set to a ClassAd float expression that you could use in the Rank attribute in a Condor job description.

**Default:** undefined
condor_requirements

Synopsis: condor_requirements = string

Description: Specify additional constraints for Condor resources. The value of condor_requirements must be a valid constraints string which is recognized by a condor_status -constraint ... command. It can reference pre-defined ClassAd attributes (like Memory, Opsys, Arch, HasJava, etc) but also custom ClassAd attributes. To define a custom attribute on a condor node, just add two lines like the ones below in the $(hostname).local config file on the node:

```bash
NORDUGRID_RESOURCE=TRUE
STARTD_EXPRS = NORDUGRID_RESOURCE, $(STARTD_EXPRS)
```

A job submitted to this resource is allowed to run on any node which satisfies the condor_requirements constraint. If condor_requirements is not set, jobs will be allowed to run on any of the nodes in the pool. When configuring multiple queues, you can differentiate them based on memory size or disk space, for example.

Default: undefined

Example:

```bash
condor_requirements=(OpSys == "linux" && NORDUGRID_RESOURCE && Memory >= 1000 && Memory < 2000)
```

sge_bin_path

Synopsis: sge_bin_path = path

Description: Path to Sun Grid Engine (SGE) binaries, Default is search for qsub command in the shell PATH

Default: undefined

Example:

```bash
sge_bin_path=/opt/n1ge6/bin/lx24-x86
```

sge_root

Synopsis: sge_root = path

Description: Path to SGE installation directory. MUST be set if SGE is used.

Default: /gridware/sge

Example:

```bash
sge_root=/opt/n1ge6
```

sge_cell

Synopsis: sge_cell = name

Description: The name of the SGE cell to use. This option is only necessary in case SGE is set up with a cell name different from ‘default’
NorduGrid ARC 6 Information, Release ARC6

Default: default
Example:

```
sge_cell=default
```

**sge_qmaster_port**

*Synopsis:* `sge_qmaster_port = port`

*Description:* The SGE port options should be used in case SGE command line clients require SGE_QMASTER_PORT and SGE_EXECD_PORT environment variables to be set. Usually they are not necessary.

Default: undefined
Example:

```
sge_qmaster_port=536
```

**sge_execd_port**

*Synopsis:* `sge_execd_port = port`

*Description:* The SGE port options should be used in case SGE command line clients require SGE_QMASTER_PORT and SGE_EXECD_PORT environment variables to be set. Usually they are not necessary.

Default: undefined
Example:

```
sge_execd_port=537
```

**sge_jobopts**

*Synopsis:* `sge_jobopts = string`

*Description:* Additional SGE options to be used when submitting jobs to SGE

Default: undefined
Example:

```
sge_jobopts=-P atlas -r yes
```

**slurm_bin_path**

*Synopsis:* `slurm_bin_path = path`

*Description:* Path to SLURM binaries, must be set if installed outside of normal PATH

Default: `/usr/bin`
Example:

```
slurm_bin_path=/usr/bin
```
slurm_wakeupperiod

Synopsis: slurm_wakeupperiod = numsec

Description: How long should infosys wait before querying SLURM for new data (seconds)
Default: 30
Example:

```
slurm_wakeupperiod=15
```

slurm_use_sacct

Synopsis: slurm_use_sacct = yes/no

Description: Indicates whether ARC should use sacct instead of scontrol to obtain information about finished jobs
Not supported by slurmpy.

Allowed values: yes, no
Default: yes
Example:

```
slurm_use_sacct=yes
```

slurm_requirements

Synopsis: slurm_requirements = string

Description: Use this option to specify extra SLURM-specific parameters.
Default: undefined
Example:

```
slurm_requirements=memory on node >> 200
```

Warning: CHANGE: new parameter proposed in Umea retreat

lsf_bin_path

Synopsis: lsf_bin_path = path

Description: The PATH to LSF bin folder
Default: /usr/bin
Example:

```
lsf_bin_path=/usr/local/lsf/bin/
```

lsf_profile_path

Synopsis: lsf_profile_path = path

Description: Path to the profile.lsf file. Infoprovider scripts will source profile.lsf to setup LSF utilities environment.
**Default**: /usr/share/lsf/conf/profile.lsf

**Example**:

```
lsf_profile_path=/usr/local/share/lsf/conf/profile.lsf
```

**lsf_architecture**

**Synopsis**: lsf_architecture = string

**Description**: CPU architecture to request when submitting jobs to LSF. Use only if you know what you are doing.

**Default**: undefined

**Example**:

```
lsf_architecture=PowerPC
```

**Warning**: CHANGE: moved here from [queue:name] block

**ll_bin_path**

**Synopsis**: ll_bin_path = path

**Description**: The PATH to the LoadLeveler bin folder

**Default**: /usr/bin

**Example**:

```
ll_bin_path=/opt/ibmll/LoadL/full/bin
```

**ll_consumable_resources**

**Synopsis**: ll_consumable_resources = yes/no

**Description**: Indicates whether the LoadLeveler setup is using Consumable Resources.

**Allowed values**: yes, no

**Default**: no

**Example**:

```
ll_consumable_resources=yes
```

**boinc_db_host**

**Synopsis**: boinc_db_host = hostname

**Description**: Connection strings for the boinc database: host

**Default**: localhost

**Example**:

```
boinc_db_host=localhost
```
boinc\_db\_port

**Synopsis:** boinc\_db\_port = port

*Description:* Connection strings for the boinc database: port

*Default:* 3306

*Example:*

```
boinc_db_port=3306
```

boinc\_db\_name

**Synopsis:** boinc\_db\_name = db\_name

*Description:* Connection strings for the boinc database: db\_name

*Default:* undefined

*Example:*

```
boinc_db_name=myproject
```

boinc\_db\_user

**Synopsis:** boinc\_db\_user = user

*Description:* Connection strings for the boinc database: db\_user

*Default:* undefined

*Example:*

```
boinc_db_user=boinc
```

boinc\_db\_pass

**Synopsis:** boinc\_db\_pass = pwd

*Description:* Connection strings for the boinc database: pwd

*Default:* undefined

*Example:*

```
boinc_db_pass=password
```

boinc\_app\_id - ID of the app handled by this CE. Setting this option makes database queries much faster in large projects with many apps.

*Default:* undefined

*Example:*

```
boinc_app_id=1
```
[lrms/ssh] block

This sub-block configures the ssh environment for remote batch management. Currently ONLY the python slurm (slurmpy) EXPERIMENTAL batch module is able to utilize the remote ssh feature. Parameters within this block are relevant if the cluster frontend is remotely located wrt. CE frontend (machine running A-REX). If specified with the parameters below, the session, cache and runtime directories will be mounted from the cluster frontend on the CE frontend using sshfs. Job submission and management will done using ssh (Paramiko).

**Warning:** CHANGE: new sub-block

remote_host

**Synopsis:** *remote_host = hostname*

**Description:** Define the remote cluster frontend machine which contains session, cache and runtime directories and where jobs should be submitted to.

**Default:** undefined

mandatory

**Example:**

remote_host=myremotehost.org

private_key

**Synopsis:** *private_key = path*

**Description:** Location of the private key which should be used establish connection to the machine specified by the ‘remote_host’ attribute.

**Default:** undefined

mandatory

**Example:**

private_key=/etc/grid-security/hostkey-priv.pem

remote_user

**Synopsis:** *remote_user = user*

**Description:** User on remote cluster which should be used for mounting directories, submitting and managing jobs, and gathering information about cluster.

**Default:** undefined

mandatory

**Example:**

remote_user=grid
remote_sessiondir

Synopsis: remote_sessiondir = path

Description: Session directory on cluster frontend to be mounted (sshfs) on CE machine at directory specified by the ‘sessiondir’ attribute in the [arex] block.

Default: undefined

Example:

remote_sessiondir=/scratch/grid

remote_cachedir

Synopsis: remote_cachedir = path

Description: Cache directory on cluster frontend to be mounted (sshfs) on CE machine at directory specified by the ‘cachedir’ attribute in the [arex] block.

Default: undefined

Example:

remote_cachedir=/scratch/cache

ssh_timeout

Synopsis: ssh_timeout = time

Description: Timeout of establishing ssh connection. Unit in seconds.

Default: 10

Example:

ssh_timeout=10

[arex] block

The [arex] block, together with its various subblocks, configures the A-REX service hosted in arched. A-REX takes care of various middleware tasks on the frontend such as job creation and management, stagein/stageout, LRMS job submission, data caching, etc...

Warning: CHANGE: renamed block

user

Synopsis: user = user[:group]

Description: Switch to a non root user/group after startup. Use with caution because of limited functionality when arex is not run under root.

Default: root

Example:
norootpower

Synopsis: norootpower = yes|no
Description: If set to yes, all job management processes will switch to mapped user’s identity while accessing session directory. This is useful if session directory is on NFS with root squashing turned on.
Allowed values: yes, no
Default: no
Example:
norootpower=yes

delegationdb

Synopsis: delegationdb = db_name
Description: specify which DB to use to store delegations. Currently supported db_names are bdb and sqlite
Default: sqlite
Example:
delegationdb=sqlite

Warning: CHANGE: modified. new default.

watchdog

Synopsis: watchdog = yes/no
Description: Specifies if additional watchdog processes is spawned to restart main process if it is stuck or dies.
Allowed values: yes, no
Default: no
Example:
watchdog=no

loglevel

Synopsis: loglevel = level
Description: (previously debug) Set loglevel of the arched daemon hosting A-REX service between 0 (FATAL) and 5 (DEBUG). Defaults to 3 (INFO).
Allowed values: 0, 1, 2, 3, 4, 5, FATAL, ERROR, WARNING, INFO, VERBOSE, DEBUG
Default: 3
Example:
loglevel=3

**Warning:** CHANGE: renamed

**logfile**

**Synopsis:** logfile = path

**Description:** Specify A-REX log file location. If using an external log rotation tool be careful to make sure it matches the path specified here.

**Default:** /var/log/arc/arex.log

**Example:**

```
logfile=/var/log/arc/arex.log
```

**Warning:** CHANGE: modified default (renamed file)

**joblog**

**Synopsis:** joblog = path

**Description:** Specifies where to store specialized log about started and finished jobs. If path is empty log is NOT written. Controlled by logrotate if default name is kept. This log is not used by any other part of ARC so can be safely disabled if you are not interested in storing jobs log.

**Default:** /var/log/arc/arex-jobs.log

**Example:**

```
joblog=""
```

**fixdirectories**

**Synopsis:** fixdirectories = yes/missing/no

**Description:** Specifies during startup A-REX should create all directories needed for it operation and set suitable default permissions. If no is specified then A-REX does nothing to prepare its operational environment. In case of missing A-REX only creates and sets permissions for directories which are not present yet. For yes all directories are created and permissions for all used directories are set to default safe values.

**Allowed values:** yes, missing, no

**Default:** yes

**Example:**

```
fixdirectories=yes
```

**controldir**

**Synopsis:** controldir = path
Description: The directory of the A-REX’s internal job metadata files. For a heavy loaded computing elements you can consider to locate controldir on a dedicated partition optimized for small random reads and writes. The directory is not needed on the nodes.

Default: /var/spool/arc/jobstatus

Example:

controldir=/var/spool/arc/jobstatus

sessiondir

Synopsis: sessiondir = path [drain]

Description: the directory which holds the sessiondirs of the grid jobs. Multiple session directories may be specified. In this case jobs are spread evenly over the session directories. If sessiondir=* is set, the session directory will be spread over the ${HOME}/.jobs directories of every locally mapped unix user. It is preferred to use common session directories. The path may be followed by drain, in which case no new jobs will be assigned to that sessiondir, but current jobs will still be processed and accessible.

This option in multivalued.

Default: /var/spool/arc/sessiondir

Example:

sessiondir=/scratch/arcsessions drain
sessiondir=*.

defaultttl

Synopsis: defaultttl = [ttl [ttr]]

Description: The ttl parameter sets the time in seconds for how long a job session directory will survive after job execution has finished. If not specified the default is 1 week. The ttr parameter sets how long information about a job will be kept after the session directory is deleted. If not specified, the ttr default is one month.

Default: 604800 2592000

Example:

defaultttl=2592000

shared_filesystem

Synopsis: shared_filesystem = yes/no

Description: Specifies if computing nodes can access folders mounted with protocols like NFS with the same pathnames as the frontend. Note that the default ‘yes’ assumes that the paths to the session directories are the same on both frontend and nodes. If these paths are not the same, then one should set the scratchdir option. The option changes the RUNTIME_NODE_SEES_FRONTEND variable in the submission scripts.

Allowed values: yes, no

Default: yes

Example:

shared_filesystem=yes
scrachdir

**Synopsis:** `scratchdir = path`

**Description:** The path on computing node to move session directory to before execution. If defined should contain the path to the directory on the computing node which can be used to store a jobs’ files during execution. Sets the environment variable `RUNTIME_LOCAL_SCRATCH_DIR`. If the variable is not set, then the session dir is not moved before execution. Don’t set this parameter unless you want to move the sessiondir to scratchdir on the node.

**Default:** undefined

**Example:**
```
scratchdir=/local/scratch/
```

shared_scratch

**Synopsis:** `shared_scratch = path`

**Description:** The 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`.

**Default:** undefined

**Example:**
```
shared_scratch=/mnt/scratch
```

tmpdir

**Synopsis:** `tmpdir = path`

**Description:** A temporary directory used by A-REX.

**Default:** `/tmp`

**Example:**
```
tmpdir=/tmp
```

runtimedir

**Synopsis:** `runtimedir = path`

**Description:** The directory which holds the additional runtime environment scripts, added by system administrator. Several directories can be specified. To enable RTEs to be advertised in the information system and used during submission the arcctl tool should be used.

This option is **multivalued**.

**Default:** undefined

**Example:**
```
runtimedir=/var/spool/arc/extraruntimes
runtimedir=/cvmfs/vo/arcruntime
```
**maxjobs**

*Synopsis:* `maxjobs = number1 number2 number3 number4 number5`

*Description:* specifies maximum allowed number of jobs. number1 - jobs which are not in FINISHED state (jobs tracked in RAM) number2 - jobs being run (SUBMITTING, INLRMS states) number3 - jobs being processed per DN number4 - jobs in whole system number5 - LRMS scripts limit (jobs in SUBMITTING and CANCELING) A parameter set to -1 means no limit.

*Default:* `-1 -1 -1 -1 -1`

*Example:*

```
maxjobs=10000 10 2000 -1 -1
```

*Warning:* CHANGE: Modified. Explicitly indicate no limit with -1. Missing number should not be allowed.

**maxrerun**

*Synopsis:* `maxrerun = number`

*Description:* Specifies how many times job can be rerun if it failed in LRMS. This is only an upper limit, the actual rerun value is set by the user in his xrsl.

*Default:* `5`

*Example:*

```
maxrerun=5
```

**statecallout**

*Synopsis:* `statecallout = state options plugin_path [plugin_arguments]`

*Description:* (previously authplugin) Enables a callout feature of A-REX: every time job goes to state A-REX will run plugin_path executable. Options consist of key=value pairs separated by comma. Possible keys are:

- `timeout` defines the timeout in seconds to wait for plugin execution (`timeout=` can be omitted).
- `onfailure`, `ontimeout`, `onsuccess` defines the action that A-REX should take on succesfull execution (exit code 0), failed execution (exit code is not 0) or execution timeout respectively.

Possible actions are:

- `pass` - continue executing job,
- `fail` - cancel job,
- `log` - write to log about the failure and continue executing job.

It is possible to use following substitutions to construct plugin command line:

- `%R` - session root (value of sessiondir in [arex] block)
- `%C` - controldir path
- `%U` - username of mapped UNIX account
- `%u` - numeric UID of mapped UNIX account
- `%g` - numeric GID of mapped UNIX account
- `%H` - home directory of mapped UNIX account as specified in `/etc/passwd`  
- `%Q` - default queue (see `lrms` configuration option in `[lrms]` block)  
- `%L` - LRMS name (see `lrms` configuration option in `[lrms]` block)  
- `%W` - ARC installation path (corresponds to the `ARC_LOCATION` environmental variable)  
- `%F` - path to configuration file for this instance  
- `%I` - job ID (substituted in runtime)  
- `%S` - job state (substituted in runtime)

Plugins included into ARC distribution:

- `arc-blahp-logger` - write accounting log for every finished job in BLAH format

This option in **multivalued**.

**Default:** undefined  
**Example:**

```
statecallout=FINISHED timeout=10, onfailure=pass
```

```
/usr/libexec/arc/arc-blahp-logger -
˓→I %I -U %u -L %C/job.%I.local -P %C/job.%I.proxy
```

**Warning:** CHANGE: renamed from authplugin

### wakeupperiod

**Synopsis:** `wakeupperiod = time`

**Description:** Specifies how often A-REX checks for new jobs arrived, job state change requests, etc. That is responsiveness of A-REX. Time is time period in seconds. Default is 3 minutes. Usually no need to change this parameter because important state changes are also triggering out-of-schedule checks. NOTE: This parameter does not affect responsiveness of backend scripts - especially `scan-LRMS-job`. That means that upper estimation of time for detecting job finished executing is sum of responsiveness of backend script + `wakeupperiod`.

**Default:** 180  
**Example:**

```
wakeupperiod=180
```

### infoproviders_timelimit

**Synopsis:** `infoproviders_timelimit = seconds`

**Description:** (previously `infoproviders_timeout`) Sets the execution time limit of the infoprovider scripts started by the A-REX. Infoprovider scripts running longer than the specified timelimit are gracefully handled by the A-REX (the behaviour depends on the state of the system) Increase this value if you have many jobs in the controldir and infoproviders need more time to process.

**Default:** 10800  
**Example:**

```
infoproviders_timelimit=10800
```

**Warning:** CHANGED: rename, `infoproviders_timelimit`, move to `[arex]` block
pidfile

Synopsis: pidfile = path
Description: Specify location of file containing PID of daemon process.
Default: /var/run/arched-arex.pid
Example:

```
pidfile=/var/run/arched-arex.pid
```

mail

Synopsis: mail = email_address
Description: Specifies the email address from where the notification mails are sent
Default: $VAR{user}@$VAR{[common]hostname}
Example:

```
mail=grid.support@somewhere.org
```

Warning: CHANGE: modified. implement a default! in the format of root@localhost.

gnu_time

Synopsis: gnu_time = path
Description: The gnu time command

Warning: CHANGE: moved to [lrms]

nodename

Synopsis: nodename = path
Description: The command to obtain hostname of computing node.

Warning: CHANGE: moved to [lrms]

helper

Synopsis: helper = user executable arguments
Description: By enabling this parameter A-REX will run an external helper program under the user user-account. The program will be kept running, every time the executable finishes it will be started again. As a limitation, currently only '.' is supported as username, which corresponds to the user running A-REX.
Default: undefined
Example:
helperlog

Synopsis: helperlog = path

Description: Configuration option to specify the location of log for helpers.

Default: /var/log/arc/job.helper.errors

Example:

```
helperlog=/var/log/arc/job.helper.errors
```

Warning: CHANGE: new parameter

forcedefaultvoms

Synopsis: forcedefaultvoms = VOMS_FQAN

Description: specify VOMS FQAN which user will be assigned if his/her credentials contain no VOMS attributes. To assign different values to different queues put this command into [queue] block.

Default: undefined

Example:

```
forcedefaultvoms=/vo/group/subgroup
```

[arex/cache] block

This subblock enables and configures the cache functionality of A-REX. A-REX can cache input files downloaded as part of stagein process of grid jobs so that subsequent jobs requiring the same file don’t have to download it again. The cached file will be symlinked (or copied) into the session directory of the job. To disable the cache functionality simply comment out the [arex/cache] config block. It is a good idea to have the cache on its own separate file system that is shared with the nodes. For more information about the cache functionality of A-REX consult the sysadmin guide.

Warning: CHANGE: new block

cachedir

Synopsis: *cachedir = cache_path [link_path]

Description: Specifies a directory to store cached data. Multiple cache directories may be specified. Cached data will be distributed evenly over the caches. Optional link_path specifies the path at which the cache_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, but copied to session directory. If a cache directory needs to be drained, then link_path should specify drain, in which case no new files will be added to the cache. Restart of arex also needed.

This option in multivalued.

Default: undefined

Example:
NorduGrid ARC 6 Information, Release ARC6

cachedir=/scratch/cache
cachedir=/shared/cache /frontend/jobcache
cachedir=/fs1/cache drain

[arex/cache/cleaner] block

This subblock enables the cleaning functionality of the cache. If this block is not enabled then the cache will not be cleaned by A-REX. Either cachesize or cachelifetime should also be set to enable cleaning.

Warning: CHANGE: new block

logfile

Synopsis: logfile = path

Description: (previously cachelogfile) sets the filename where output of the cache-clean tool should be logged. Defaults to /var/log/arc/cache-clean.log.

Default: /var/log/arc/cache-clean.log

Example:
logfile=/tmp/cache-clean.log

Warning: CHANGE: renamed as logfile

loglevel

Synopsis: loglevel = level

Description: (previously cacheloglevel) specifies the level of logging by the cache-clean tool, between 0 (FATAL) and 5 (DEBUG). Defaults to 3 (INFO).

Allowed values: 0, 1, 2, 3, 4, 5, FATAL, ERROR, WARNING, INFO, VERBOSE, DEBUG

Default: 3

Example:
loglevel=4

Warning: CHANGE: renamed as loglevel

cachesize

Synopsis: cachesize = max min

Description: Specifies high and low watermarks for space used by cache, as a percentage of the space on the file system on which the cache directory is located. 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 the cache on its own separate file system.

Default: 100 100

Example:
calculatesize

Synopsis: calculatesize = filesystem/cachedir

Description: (previously cacheshared) specifies the way the space occupied by the cache will be calculated. If set to cachedir then cache-clean calculates the size of the cache instead of using filesystem used space.

Allowed values: filesystem, cachedir

Default: filesystem

Example:

`calculatesize=cachedir`

Warning: CHANGE: renamed parameter
cachelifetime

Synopsis: cachelifetime = time

Description: Turns on time-based file cleaning. Files accessed less recently than the given time period will be deleted. Example values of this option are 1800, 90s, 24h, 30d. When no suffix is given the unit is seconds.

Default: undefined

Example:

`cachelifetime=30d`
cachespacetool

Synopsis: cashespacetool = path [options]

Description: specifies an alternative tool to df that cache-clean should use to obtain space information on the cache file system. The output of this command must be total_bytes used_bytes. The cache directory is passed as the last argument to this command.

Default: undefined

Example:

`cachespacetool=/etc/getspace.sh`
cacheCLEANtimeout

Synopsis: cacheCLEANtimeout = time

Description: 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).

Default: 3600

Example:
[arex/data-staging] block

This subblock enables and configures the data staging capabilities of A-REX. A subsystem called DTR (Data Transfer Reloaded) is responsible for collecting input data for a job before submission to the LRMS, and for staging out data after the job has finished. Automagic data staging of A-REX is a very powerful feature, disabling this functionality (by commenting out the subblock) is not recommended.

**Warning:** CHANGE: RENAMED block

### loglevel

**Synopsis:** loglevel = number

**Description:** (previously debug) Sets the log level for transfer logging in job.id.errors files, between 0 (FATAL) and 5 (DEBUG). Default is to use value set by loglevel option in [arex] section.

**Allowed values:** 0, 1, 2, 3, 4, 5, FATAL, ERROR, WARNING, INFO, VERBOSE, DEBUG

**Default:** $VAR{[arex]loglevel}

**Example:**

```
loglevel=4
```

**Warning:** CHANGE: renamed as loglevel.

### logfile

**Synopsis:** logfile = path

**Description:** (previously central_logfile) A central file in which all data staging messages from every job will be collected and logged in addition to their job.id.errors files. If this option is not present or the path is empty the log file is not created. This file is not automatically controlled by logrotate unless you name it as /var/log/arc/datastaging.log.

**Default:** undefined

**Example:**

```
logfile=/var/log/arc/datastaging.log
```

**Warning:** CHANGE: renamed

### statefile

**Synopsis:** statefile = path

**Description:** (previously dtrlog) A file in which data staging state information (for monitoring and recovery purposes) is periodically dumped.

**Default:** $VAR{[arex]controldir}/dtr.state
Example:

statefile=/tmp/dtr.state

**Warning:** CHANGE: renamed, modified: new default value

**usehostcert**

*Synopsis:* usehostcert = yes/no

*Description:* Whether the A-REX host certificate should be used for communication with remote hosts instead of the users’ proxies.

*Allowed values:* yes, no

*Default:* no

*Example:*

```
usehostcert=yes
```

**maxtransfertries**

*Synopsis:* maxtransfertries = number

*Description:* the maximum number of times download and upload will be attempted per job (retries are only performed if an error is judged to be temporary)

*Default:* 10

*Example:*

```
maxtransfertries=20
```

**passivetransfer**

*Synopsis:* passivetransfer = yes/no

*Description:* If yes, gridftp transfers are passive. Setting this option to yes can solve transfer problems caused by firewalls.

*Allowed values:* yes, no

*Default:* yes

*Example:*

```
passivetransfer=yes
```

**Warning:** CHANGE: new default: yes

**globus_tcp_port_range**

*Synopsis:* globus_tcp_port_range = port_range
**Description:** In a firewalled environment the software which uses GSI needs to know what ports are available. This parameter is only needed if `passivetransfer=no` was set. These variables are similar to the Globus environment variables `GLOBUS_TCP_PORT_RANGE` and `GLOBUS_UDP_PORT_RANGE`.

**Default:** 9000,9300

**Example:**

```
globus_tcp_port_range=9000,12000
```

**Warning:** CHANGE: moved here

---

**globus_udp_port_range**

**Synopsis:** `globus_udp_port_range = port_range`

**Description:** In a firewalled environment the software which uses GSI needs to know what ports are available. This parameter is only needed if `passivetransfer=no` was set. These variables are similar to the Globus environment variables `GLOBUS_TCP_PORT_RANGE` and `GLOBUS_UDP_PORT_RANGE`.

**Default:** 9000,9300

**Example:**

```
globus_udp_port_range=9000,12000
```

**Warning:** CHANGE: moved here

---

**httpgetpartial**

**Synopsis:** `httpgetpartial = yes/no`

**Description:** If yes, HTTP GET transfers may transfer data in chunks/parts. If no - data is always transferred in one piece.

**Allowed values:** yes, no

**Default:** no

**Example:**

```
httpgetpartial=no
```

**Warning:** CHANGE: new default: no

---

**speedcontrol**

**Synopsis:** `speedcontrol = min_speed min_time min_average_speed max_inactivity`

**Description:** specifies how slow data transfer must be to trigger error. Transfer is cancelled if speed is below `min_speed` bytes per second for at least `min_time` seconds, or if average rate is below `min_average_speed` bytes per second, or no data was transferred for longer than `max_inactivity` seconds. Value of zero turns feature off.

**Default:** 0 300 0 300

**Example:**

```
speedcontrol=0 300 0 300
```
speedcontrol=0 300 100 300
speedcontrol=""

**Warning:**  CHANGE: modified: missing parameter `speedcontrol=` should also turn the feature off, not just zero value.

### maxdelivery

**Synopsis:** maxdelivery = number

**Description:** Maximum number of concurrent file transfers, i.e. active transfers using network bandwidth. This is the total number for the whole system including any remote staging hosts.

**Default:** 10

**Example:**

```
maxdelivery=40
```

### maxprocessor

**Synopsis:** maxprocessor = number

**Description:** Maximum number of concurrent files in each of the DTR internal pre- and post-processing states, eg cache check or replica resolution.

**Default:** 10

**Example:**

```
maxprocessor=20
```

### maxemergency

**Synopsis:** maxemergency = number

**Description:** Maximum emergency slots which can be assigned to transfer shares when all slots up to the limits configured by the above two options are used by other shares. This ensures shares cannot be blocked by others.

**Default:** 1

**Example:**

```
maxemergency=5
```

### maxprepared

**Synopsis:** maxprepared = number

**Description:** Maximum number of files in a prepared state, i.e. pinned on a remote storage such as SRM for transfer. A good value is a small multiple of maxdelivery.

**Default:** 200

**Example:**
maxprepared=250

sharepolicy

**Synopsis:** sharepolicy = grouping

**Description:** (previously sharetype) Defines the mechanism to be used for the grouping of the job transfers. DTR assigns the transfers to shares, so that those shares can be assigned to different priorities. Possible values for grouping are dn, voms:vo, voms:role and voms:group:

- **dn** each job is assigned to a share based on the DN of the user submitting the job.
- **voms:vo** each job is assigned to a share based on the VO specified in the proxy.
- **voms:role** each job is assigned to a share based on the role specified in the first attribute found in the proxy.
- **voms:group** each job is assigned to a share based on the group specified in the first attribute found in the proxy.

In case of the voms schemes, if the proxy is not a VOMS proxy, then a default share is used. If sharepolicy is not set then the client-defined priority is applied.

**Default:** undefined

**Example:**

sharepolicy=voms:role

**Warning:** CHANGE: renamed

sharepriority

**Synopsis:** sharepriority = share priority

**Description:** (previously definedshare) Defines a share with a fixed priority, different from the default (50). Priority is an integer between 1 (lowest) and 100 (highest).

This option in multivalued.

**Default:** undefined

**Example:**

sharepriority=myvo:students 20
sharepriority=myvo:production 80

**Warning:** CHANGE: renamed

copyurl

**Synopsis:** copyurl = url_head local_path

**Description:** Configures that DTR should use copy instead of download in case of certain stagein files. URLs, starting from ‘url_head’ should be accessed in a different way (most probably unix open). The ‘url_head’ part of the URL will be replaced with ‘local_path’ and file from obtained path will be copied to the session directory.

NOTE: ‘local_path’ can also be of URL type.
This option in **multivalued**.

**Default**: undefined

**Example**:

```plaintext
copyurl=gsiftp://example.org:2811/data/ /data/
copyurl=gsiftp://example2.org:2811/data/ /data/
```

**linkurl**

**Synopsis**: `linkurl = url_head local_path [node_path]`

**Description**: Identical to ‘copyurl’, configures DTR so that for certain URLs files won’t be downloaded or copied (in case of copyurl), but soft-link will be created. The ‘local_path’ 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.

This option in **multivalued**.

**Default**: undefined

**Example**:

```plaintext
linkurl=gsiftp://somewhere.org/data /data
linkurl=gsiftp://example.org:2811/data/ /scratch/data/
```

**use_remote_acix**

**Synopsis**: `use_remote_acix = URL`

**Description**: (previously acix_endpoint) If configured then the ARC Cache Index, available at the URL, will be queried for every input file specified in a job description and any replicas found in sites with accessible caches will be added to the replica list of the input file. The replicas will be tried in the order specified by preferredpattern variable.

**Default**: undefined

**Example**:

```plaintext
use_remote_acix=https://cacheindex.ndgf.org:6443/data/index
```

**Warning**: CHANGE: rename

**preferredpattern**

**Synopsis**: `preferredpattern = pattern`

**Description**: specifies a preferred pattern on which to sort multiple replicas of an input file. It consists of one or more patterns separated by a pipe character (|) listed in order of preference. Replicas will be 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. If an exclamation mark (!) is used at the beginning of a pattern, any replicas matching the pattern will be excluded from the sorted replicas.

**Default**: undefined

**Example**: 

```plaintext
```
The following options are used to configure multi-host data staging deployment scenario. In that setup a couple of additional data staging boxes are enabled to off-load data transfers.

**deliveryservice**

*Synopsis:* deliveryservice = URL

*Description:* The URL to a remote data delivery service which can perform remote data staging.

*Default:* undefined

*Example:*

deliveryservice=https://myhost.org:443/datadeliveryservice

**localdelivery**

*Synopsis:* localdelivery = yes/no

*Description:* If any deliveryservice is defined, this option determines whether local data transfer is also performed.

*Allowed values:* yes, no

*Default:* no

*Example:*

localdelivery=yes

**remotesizelimit**

*Synopsis:* remotesizelimit = size

*Description:* Lower limit on file size (in bytes) of files that remote hosts should transfer. Can be used to increase performance by transferring small files using local processes.

*Default:* undefined

*Example:*

remotesizelimit=100000

**[arex/ws] block**

A-REX exposes a set of Web Service interfaces that can be used to create and manage jobs, obtain information about the CE and the jobs, handle delegations, access cache information, so on. Comment out this block if you don’t want to provide WS-interfaces for various A-REX functionalities.

*Warning:* CHANGE: new block. most of the parameters originates from the old [grid-manager] block
wsurl

Synopsis: wsurl = url

Description: (previously arex_mount_point) Specifies the base URL under which the web service interfaces will be available. The URL argument must be a full URL consisting of protocol+host+port+path: e.g. https://<hostname>:<port>/<path> Make sure the chosen port is not blocked by firewall or other security rules.

Default: https://$VAR{{common}hostname}:443/arex

Example: wsurl=https://piff.hep.lu.se:443/arex


logfile

Synopsis: logfile = path

Description: (previously wslogfile) Specify log file location for WS-interface operations.

Default: /var/log/arc/ws-interface.log

Example: logfile=/var/log/arc/ws-interface.log

Warning: CHANGE: renamed.

max_job_control_requests

Synopsis: max_job_control_requests = number

Description: The max number of simultaneously processed job management requests over WS interface - like job submission, cancel, status check etc.

Default: 100

Example: max_job_control_requests=100

max_infosys_requests

Synopsis: max_infosys_requests = number

Description: The max number of simultaneously processed info requests over WS interface.

Default: 1

Example: max_infosys_requests=1
max_data_transfer_requests

Synopsis: max_data_transfer_requests = number

Description: The max number of simultaneously processed data transfer requests over WS interface - like data staging.

Default: 100

Example:

max_data_transfer_requests=100

[arex/ws/jobs] block

AREX offers a set of web service interfaces implemted via either REST or WS-SOAP. This block enables the job management, info query, delegation protocols through both the REST and EMIES interface. Consult the http://www.nordugrid.org/documents/EMI-ES-Specification_v1.16.odt for EMIES inteface description and read the TO-BE-PROVIDED ARC-REST interface specification.

Warning: CHANGE: new subblock

allownew

Synopsis: allownew = yes/no

Description: The ‘allownew’ config parameter sets if the Computing Element accepts submission of new jobs via the WS-interface. This parameter can be used to close down the CE.

Allowed values: yes, no

Default: yes

Example:

allownew=yes

Warning: CHANGE: new parameter in this block. Implement support in the code for WS-interface.

allownew_override

Synopsis: allownew_override = [authgroup ...]

Description: (previously allowsubmit) Defines which authorization groups are allowed to submit new jobs via the WS-interfaces when the CE is closed with allownew=no. Note that it requires the allownew=no to be set.

This option in multivalued.

Default: undefined

Example:

allownew_override=biousers atlasusers
allownew_override=yourauthgroup
**allowaccess**

**Synopsis:** allowaccess = authgroup

**Description:** (previously groupcfg) Defines that the specified authgroup members are authorized to access the ARC-CE via this interface. A related config option the denyaccess (see below) can be used to reject access. Multiple allowaccess and denyaccess authorization statements are allowed within a configuration block. These statements are processed sequentially in the order they are specified in the config block. The processing stops on first allowaccess or denyaccess statement matching the authgroup membership. If there are no authorization statements specified, then no additional restrictions are applied for authorizing user access and the interface is open to everybody authenticated.

**Default:** undefined

This option in **multivalued**.

**Example:**

```
allowaccess=biouers
allowaccess=atlusers
```

**denyaccess**

**Synopsis:** denyaccess = authgroup

**Description:** Defines that the specified authgroup members are REJECTED, not authorized to access the ARC-CE via this interface. Note that a related config option the allowaccess (see above) can be used to grant access. Multiple denyaccess and allowaccess authorization statements are allowed within a configuration block. These statements are processed sequentially in the order they are specified in the config block. The processing stops on first allowaccess or denyaccess statement matching the authgroup membership. If there are no authorization statements specified, then no additional restrictions are applied for authorizing user access and the interface is open to everybody authenticated.

**Default:** undefined

This option in **multivalued**.

**Example:**

```
denyaccess=blacklisted-users
```

**maxjobdesc**

**Synopsis:** maxjobdesc = size

**Description:** specifies maximal allowed size of job description in bytes. Default value is 5MB. Use 0 to set unlimited size.

**Default:** 5242880

**Example:**

```
maxjobdesc=0
```

**Warning:** CHANGE: new parameter in this block. Implement support in the code for WS-interface.
[arex/ws/cache] block

The content of the A-REX cache can be accessed via a WS-interface. Configuring this block will allow reading cache files through a special URL. For example, if the remote file gsiftp://remotehost/file1 is stored in the cache and the WS interfaces (configured above) are available via wsurl of https://hostname:443/arex/, then the cached copy of the file can be access via the following special URL: https://hostname:443/arex/cache/gsiftp://remotehost/file1

Comment out this block if you don’t want to expose the cache content via WS-interface.

**Warning:** CHANGE: new block

### cacheaccess

**Synopsis:** cacheaccess = rule

**Description:** This parameter defines the access control rules for the cache wsinterface, the rules for allowing access to files in the cache remotely through the A-REX web interface. If not set, then no one can access anything. The default is not set that means complete denial. A rule has three parts:

1. Regular expression defining a URL pattern
2. Credential attribute to match against a client’s credential
3. Regular expression defining a credential value to match against a client’s credential

A client is allowed to access the cached file if a URL pattern matches the cached file URL and the client’s credential has the attribute and matches the value required for that pattern. Possible values for credential attribute are dn, voms:vo, voms:role and voms:group.

This option in **multivalued**.

**Default:** undefined

**Example**:

```plaintext
cacheaccess=gsiftp://host.org/private/data/.* voms:vo myvo:production
cacheaccess=gsiftp://host.org/private/data/bob/.* dn /O=Grid/O=NorduGrid/.*
```

[arex/ws/candypond] block

The CandyPond (Cache and deliver your pilot on-demand data) A-REX Web Service (previously called Cache Service) exposes various useful data-staging related operations for the pilot job model where input data for jobs is not known until the job is running on the worker node. This service is intended to be used by A-REX managed jobs. This service requires the [arex/data-staging] functionality.

The CandyPond service is available via the wsurl/candypond URL (e.g. https://hostname:443/arex/candypond)

**Warning:** CHANGE: NEW block, renamed service

[arex/ws/argus] block

The Web Service components of A-REX may directly use the Argus service (https://twiki.cern.ch/twiki/bin/view/EGEE/AuthorizationFramework) for requesting authorization decisions and performing client mapping to a local user account. This block turns on and configures the A-REX WS - Argus integration. When this block is enabled, A-REX will communicate to Argus PEP or PDP service for every WS interface operation! Comment out this block if you don’t intend to use any external Argus service with A-REX WS interfaces. Using Argus with gridftp interface is possible only via LCMAPS callout.
**Warning:** CHANGE: new block for the Argus functionality. Change code so that Argus is turned on/off based on this block

---

**arguspep_endpoint**

*Synopsis:* \*arguspep_endpoint = url

*Description:* Specifies URL of Argus PEPD service to use for authorization and user mapping. It is worth to mention that `requireClientCertAuthentication` (default is false) item of pepd.ini (configuration of Argus PEPD service) is set to be ‘true’, then https should be used, otherwise http is proper.

*Note:* Argus will be contacted for every WS interface operation requested!

*Default:* undefined

*Example:*

```
arguspep_endpoint=https://somehost.somedomain:8154/authz
```

---

**arguspep_profile**

*Synopsis:* arguspep_profile = profile_name

*Description:* Defines which communication profile to use while communicating with Argus PEPD service. Possible values for profile_name are:

- direct - pass all authorization attributes (only for debugging)
- subject - pass only subject name of client
- emi - ARC native profile developed in EMI project. This is default option.

*Allowed values:* direct, subject, emi

*Default:* emi

*Example:*

```
arguspep_profile=emi
```

---

**arguspep_usermap**

*Synopsis:* arguspep_usermap = yes/no

*Description:* Specifies whether response from Argus service may define mapping of client to local account. Default is ‘no’. Note that Argus is contacted after all the other user mapping is performed. Hence it can overwrite all other decisions.

*Allowed values:* yes, no

*Default:* no

*Example:*

```
arguspep_usermap=no
```
arguspdp_endpoint

Synopsis:  arguspdp_endpoint = url

Description: Specifies URL of Argus PDP service to use for authorization and user mapping. It is worth to mention that requireClientCertAuthentication (default is false) item of pdp.ini (configuration of Argus PDP service) is set to be 'true', then https should be used, otherwise http is proper.

Note: Argus will be contacted for every WS interface operation requested!

Default: undefined

Example:

arguspdp_endpoint=https://somehost.somedomain:8152/authz

arguspdp_profile

Synopsis:  arguspdp_profile = profile_name

Description: Defines which communication profile to use while communicating with Argus PDP service. Possible values for profile_name are:

   subject - pass only subject name of client
   emi - ARC native profile developed in EMI project. This is default option.

Allowed values: subject, emi

Default: emi

Example:

arguspdp_profile=emi

Warning: CHANGE: modified parameter values: remove cream (or at least hide from the arc.conf.reference)

arguspdp_acceptnotapplicable

Synopsis:  arguspdp_acceptnotapplicable = yes/no

Description: Specify if the NotApplicable decision returned by Argus PDP service is treated as reason to deny request. Default is 'no', which treats NotApplicable as reason to deny request.

Allowed values: yes, no

Default: no

Example:

arguspdp_acceptnotapplicable=no

[arex/jura] block

JURA is the accounting record generating and reporting module of A-REX. A-REX periodically executes JURA to processes the job log files, and based on the accounting target destinations specified in them, JURA creates usage records in the appropriate format and sends the generated records to one or more accounting service destinations.
Optionally, the generated accounting records can be archived on the CE. Enable and configure this block if you want to send accounting records to accounting services. Note that a dedicated `accounting target subblock` is needed for every accounting destination. The target subblocks are either of a type `apel` or `sgas`: `[arex/jura/apel:targetname]` or `[arex/jura/sgas:targetname]`

**Warning:** CHANGE: new dedicated block for JURA

**Warning:** CHANGE: A-REX should NOT provide the possibility of client-side target selection (in xrsI) any longer!

---

### logfile

**Synopsis:** `logfile = path`

**Description:** (previously jobreport_logfile) The name of the jura logfile.

**Default:** `/var/log/arc/jura.log`

**Example:**

```bash
logfile=/var/log/arc/jura.log
```

**Warning:** CHANGE: renamed.

---

### loglevel

**Synopsis:** `loglevel = number`

**Description:** Log level for the JURA accounting module.

**Allowed values:** 0, 1, 2, 3, 4, 5, Fatal, Error, Warning, Info, Verbose, Debug

**Default:** 3

**Example:**

```bash
loglevel=3
```

**Warning:** CHANGE: new parameter! implement it in the code

---

### vomsless_vo

**Synopsis:** `vomsless_vo = voname[#voissuer]`

**Description:** This parameter allows the sysadmin to manually assign VOs to jobs that were submitted with VOMS-less grid proxies. voname is the VO name to be used in the generated records (the same as expected in voms-proxy) optional voissuer (relevant to SGAS only) value is the VOMS server identity (certificate DN).

**Default:** undefined

**Example:**
**vo_group**

*Synopsis:* `vo_group = group`

*Description:* Adds an additional VO group attribute to the usage records.

*Default:* undefined

*Example:*

```
vo_group=/atlas/production
```

**urdelivery_keepfailed**

*Synopsis:* `urdelivery_keepfailed = days`

*Description:* Specifies for how many days JURA will try to send a record to the destination accounting service before it gives up. Records not successfully sent by after the number of days expired will be deleted from the controldir/logs directory. The deleted records are nevertheless archived if archiving was turned on. This parameter can be overwritten with a target specific value.

*Default:* 30

*Example:*

```
urdelivery_keepfailed=30
```

**urdelivery_frequency**

*Synopsis:* `urdelivery_frequency = seconds`

*Description:* (previously `jobreport_period`) Specifies the frequency of message publishing: how often the JURA process is started by A-REX.

*Default:* 3600

*Example:*

```
urdelivery_frequency=3600
```
x509_host_key

Synopsis: x509_host_key = path

Description: Optional parameter to overwrite [common] block values.

Default: $VAR[[common]x509_host_key]

Example:

x509_host_key=/etc/grid-security/hostkey.pem

Warning: CHANGE: new parameter in this block

x509_host_cert

Synopsis: x509_host_cert = path

Description: Optional parameter to overwrite [common] block values.

Default: $VAR[[common]x509_host_cert]

Example:

x509_host_cert=/etc/grid-security/hostcert.pem

Warning: CHANGE: new parameter in this block

x509_cert_dir

Synopsis: x509_cert_dir = path

Description: Optional parameter to overwrite [common] block values.

Default: $VAR[[common]x509_cert_dir]

Example:

x509_cert_dir=/etc/grid-security/certificates

Warning: CHANGE: new parameter in this block

[arex/jura/archiving] block

This block enables the archiving functionality to manage local accounting database.

When enabled the usage records generated by JURA are stored in the specified archivedir directory on the disk.

Stored records are then processed by accounting archive manager that process the records, store data into SQLite accounting database, create new archive structure and maintains the accounting database aging.

By default the archiving is turned off.
archivedir

**Synopsis:** archivedir = path

**Description:** Sets the directory path for the jura archived usage records. Usage records generated by jura reporter are stored in this directory with the following file naming convention used: `usagerecord<Type>.<jobid>.<random>`. Here Type is one of the supported record formats such as OGFUR or CAR.

**Default:** /var/spool/arc/jura/archive

**Example:**
```
archivedir=/var/spool/arc/jura/archive
```

logfile

**Synopsis:** logfile = path

**Description:** The name of the accounting database manager logfile.

**Default:** /var/log/arc/jura-archive-manager.log

**Example:**
```
logfile=/var/log/arc/jura-archive-manager.log
```

archivettl

**Synopsis:** archivettl = days

**Description:** The records time to live (ttl) parameter sets the number of days to keep the usage record files in the accounting database If not specified the files are kept forever.

**Default:** undefined

**Example:**
```
archivettl=365
```

**Warning:** CHANGE: new parameter. new feature to implement!

[arex/jura/sgas:targetname] block

An SGAS sub-block of [arex/jura] enables and configures an SGAS accounting server as a target destination to which JURA will send properly formatted usage records. The [arex/jura] config values are applicable for every target sub-block but also can be overwritten. You need to define a separate block with a unique targetname for every SGAS target server.
Warning: CHANGE: new dedicated sub-blocks for SGAS JURA targets

targeturl

Synopsis: *targeturl = url

Description: The service endpoint URL of SGAS server.

Default: undefined

Example:

targeturl=https://grid.uio.no:8001/Logger

Warning: CHANGE: new parameter

localid_prefix

Synopsis: localid_prefix = prefix_string

Description: Sets a prefix value for the LocalJobID ur parameter for the SGAS usage records.

Default: undefined

Example:

localid_prefix=some_text_for_SGAS

Warning: CHANGE: new parameter

vofilter

Synopsis: vofilter = vo

Description: Configures a job record filtering mechanism based on the VO attribute of the jobs. Only the matching job records, which was one of VO that you set here, will be sent to the target accounting service.

This option in multivalued.

Default: undefined

Example:

vofilter=atlas
vofilter=fgi.csc.fi

Warning: CHANGE: new parameter

urbatchsize

Synopsis: urbatchsize = number
NorduGrid ARC 6 Information, Release ARC6

*Description:* JURA sends usage records not one-by-one, but in batches. This option sets the size of a batch. Zero value means unlimited batch size. This option can also be set in the sgas/apel target blocks to overwrite common value.

*Default:* 50

*Example:*

```
urbatchsize=80
```

**Warning:** CHANGE: new parameter.

### [arex/jura/apel:targetname] block

An APEL sub-block of [arex/jura] enables and configures an APEL accounting server as a target destination to which JURA will send properly formatted usage records. The [arex/jura] config values are applicable for every target sub-block but also can be overwritten. You need to define a separate block with a unique targetname for every APEL target server.

**Warning:** CHANGE: new dedicated sub-blocks for APEL JURA targets

### targeturl

**Synopsis:** *targeturl = url*

*Description:* The service endpoint URL of the APEL accounting server.

*Default:* undefined

*Example:*

```
targeturl=https://mq.cro-ngi.hr:6162
```

**Warning:** CHANGE: new parameter

### topic

**Synopsis:** topic = topic_name

*Description:* Sets the name of the APEL topic to which JURA will publish the accounting records.

*Default:* /queue/global.accounting.cpu.central

*Example:*

```
topic=/queue/global.accounting.test.cpu.central
```

**Warning:** CHANGE: new parameter.
**gocdb_name**

*Synopsis:* \*gocdb\_name = name

*Description:* Can be used to specify the GOCDB name of the resource. This value would be seen as Site attribute in the generated APEL record.

*Default:* undefined

*Example:*

```
gocdb_name=GRID_UIO_NO
```

**Warning:** CHANGE: new parameter.

**benchmark_type**

*Synopsis:* benchmark\_type = type

*Description:* Type of benchmark (Si2k, Sf2k, HEPSPEC) to be reported in every UR.

*Default:* undefined

*Example:*

```
benchmark_type=HEPSPEC
```

**Warning:** CHANGE: new parameter

**benchmark_value**

*Synopsis:* benchmark\_value = number

*Description:* The value of the benchmark to be reported in every UR.

*Default:* undefined

*Example:*

```
benchmark_value=2.4
```

**Warning:** CHANGE: new parameter

**benchmark_description**

*Synopsis:* benchmark\_description = string

*Description:* Additional description for the benchmark to be reported in every UR.

*Default:* undefined

*Example:*

```
benchmark_description=some description for benchmark
```
**use_ssl**

*Synopsis:* use_ssl = yes/no

*Description:* Turns on/off ssl for the SSM communication with APEL server.

*Allowed values:* yes, no

*Default:* no

*Example:*

```
use_ssl=yes
```
[arex/ganglia] block

This block enables the monitoring of ARC-specific metrics. Earlier versions (ARC < 6.0) relied only on the standalone tool gangliarc, ganglia is now instead integrated into ARC, and gangliarc is obsolete. Note that AREX ganglia (as gangliarc did) depends on an existing ganglia installation, as it sends its metrics to a running gmond process.

Warning: CHANGE: RENAMED block

gmetric_bin_path

Synopsis: gmetric_bin_path = path

Description: (previously gmetric_exec) The path to gmetric executable.

Default: /usr/bin/gmetric

Example:

```
gmetric_bin_path=/usr/local/bin/gmetric
```

Warning: CHANGE: Moved from deleted [gangliarc] block and renamed. Path used in arc-ganglia implementation JobsMetrics.cpp

metrics

Synopsis: metrics = name_of_the_metrics

Description: the metrics to be monitored. metrics takes a comma-separated list of one or more of the following metrics: - staging – number of tasks in different data staging states - not yet implemented - cache – free cache space - session – free session directory space - heartbeat – last modification time of A-REX heartbeat - processingjobs – the number of jobs currently being processed by ARC (jobs between PREPARING and FINISHING states) - not yet implemented

• failedjobs – the number of failed jobs per last 100 finished
• jobstates – number of jobs in different A-REX stages
• all – all of the above metrics

Default: all

Allowed values: staging, cache, session, heartbeat, processingjobs, failedjobs, jobstates, all

Example:

```
metrics=all
```

frequency

Synopsis: frequency = seconds

Description: The period between each information gathering cycle, in seconds.

Default: 60

Example:
Warning: CHANGE: default increased from 20s to one minute

[gridftp] block

This block enables and configures the gridftp server. The usage of the gridftp is twofold in connection with ARC:
1) The server together with its custom jobplugin can be used as a job submission and management interface for
an ARC CE. 2) The server with the filedirplugin can be used as a very simplistic storage element. This block
configures the common server capabilities. To make the gridftp service functional, you need to enable at least one
of the plugin subblocks as well.

user

Synopsis: user = user[:group]

Description: Switch to a non root user/group after startup WARNING: Make sure that the certificate files are
owned by the user/group specified by this option.

Default: root:root

Example:

user=grid

loglevel

Synopsis: loglevel = level

Description: (previously debug) Set log level of the gridftp daemon, between 0 (FATAL) and 5 (DEBUG).
Default is 3 (INFO).

Allowed values: 0, 1, 2, 3, 4, 5, FATAL, ERROR, WARNING, INFO, VERBOSE, DEBUG

Default: 3

Example:

loglevel=2

Warning: CHANGE: renamed

logfile

Synopsis: logfile = path

Description: Set logfile location of the gridftp server.

Default: /var/log/arc/gridftpd.log

Example:

logfile=/var/log/arc/gridftpd.log
**pidfile**

*Synopsis:* pidfile = path

*Description:* Specify location of file containing PID of daemon process.

*Default:* /var/run/gridftpd.pid

*Example:*

```
pidfile=/var/run/gridftpd.pid
```

**port**

*Synopsis:* port = bindport

*Description:* Port to listen on. For gridftp-based job submission strongly advised to use the default 2811 port because 3rd party clients assume ARC CE using that port.

*Default:* 2811

*Example:*

```
port=2811
```

**allowencryption**

*Synopsis:* allowencryption = yes/no

*Description:* (previously encryption) should data encryption be allowed on client request. Encryption is very heavy, therefore the default is no.

*Allowed values:* yes, no

*Default:* no

*Example:*

```
allowencryption=no
```

**Warning:** CHANGE: RENAMED as allowencryption

**allowactivedata**

*Synopsis:* allowactivedata = yes/no

*Description:* if no, only passive data transfer is allowed. By default both passive and active data transfers are allowed.

*Default:* yes

*Example:*

```
allowactivedata=yes
```
maxconnections

Synopsis: maxconnections = number

Description: The maximum number of connections accepted by a gridftpd server.

Default: 100

Example:

```
maxconnections=200
```

defaultbuffer

Synopsis: defaultbuffer = size

Description: Defines size of every buffer for data reading/writing. The actual value may decrease if the cumulative size of all buffers exceeds value specified by maxbuffer.

Default: 65536

Example:

```
defaultbuffer=65536
```

maxbuffer

Synopsis: maxbuffer = size

Description: Defines maximal amount of memory in bytes to be allocated for all data reading/writing buffers. Default is 640kB. The number of buffers is \( \max \{3, \min \{41, 2P + 1\}\}\), where \(P\) is the parallelism level requested by the client. Hence, even without parallel streams enabled number of buffers will be 3.

Default: 655360

Example:

```
maxbuffer=655360
```

globus_tcp_port_range

Synopsis: globus_tcp_port_range = port_range

Description: In a firewalled environment the software which uses GSI needs to know what ports are available. If not set a random port is selected. These variable are similar to the Globus environment variables: GLOBUS_TCP_PORT_RANGE and GLOBUS_UDP_PORT_RANGE.

Default: 9000,9300

Example:

```
globus_tcp_port_range=9000,12000
```

globus_udp_port_range

Synopsis: globus_udp_port_range = port_range

Description: In a firewalled environment the software which uses GSI needs to know what ports are available. If not set a random port is selected. These variable are similar to the Globus environment variables: GLOBUS_TCP_PORT_RANGE and GLOBUS_UDP_PORT_RANGE.
Default: 9000, 9300
Example:

```
globus_udp_port_range=9000,12000
```

**firewall**

**Synopsis:** firewall = hostname

**Description:** The hostname or IP address to use in response to PASV command instead of the IP address of a network interface of computer.

Default: undefined

Example:

```
firewall=my.host.org
```

**x509_host_key**

**Synopsis:** x509_host_key = path

**Description:** Optional parameter to overwrite [common] block values.

Default: $VAR{[common]x509_host_key}

Example:

```
x509_host_key=/etc/grid-security/hostkey.pem
```

**Warning:** CHANGE: renamed

**x509_host_cert**

**Synopsis:** x509_host_cert = path

**Description:** Optional parameter to overwrite [common] block values.

Default: $VAR{[common]x509_host_cert}

Example:

```
x509_host_cert=/etc/grid-security/hostcert.pem
```

**Warning:** CHANGE: renamed

**x509_cert_dir**

**Synopsis:** x509_cert_dir = path

**Description:** Optional parameter to overwrite [common] block values.

Default: $VAR{[common]x509_cert_dir}

Example:
[gridftp/jobs] block

The jobplugin of the gridftp server implements a custom job management and submission interface of ARC CE. This subblock enables and configures that interface. Consult the Technical Reference within the ARC sysadmin guide for the interface specification. Comment out this subblock if you don’t want a gridftp-based job interface.

allownew

**Synopsis:** allownew = yes/no

**Description:** This parameter sets if the ARC CE accepts submission of new jobs via the gridftp interface. This parameter can be used to close down the ARC CE.

**Allowed values:** yes, no

**Default:** yes

**Example:**

```
allownew=yes
```

allownew_override

**Synopsis:** allownew_override = [authgroup ...]

**Description:** (previously allowsubmit) Defines which authorization groups are allowed to submit new jobs via the gridftp interface when the CE is closed with allownew=no. Note that it requires the allownew=no to be set.

This option in **multivalued**.

**Default:** undefined

**Example:**

```
allownew_override=biousers atlasusers
allownew_override=yourauthgroup
```

**Warning:** CHANGE: rename, modify behaviour: possible to specify several groups on a single line!

allowaccess

**Synopsis:** allowaccess = authgroup

**Description:** (previously groupcfg) Defines that the specified authgroup members are authorized to access the ARC-CE via this interface. A related config option the denyaccess (see below) can be used to reject access. Multiple allowaccess and denyaccess authorization statements are allowed within a configuration block. These statements are processed sequentially in the order they are specified in the config block. The processing stops on first allowaccess or denyaccess statement matching the authgroup membership. If there are no authorization statements specified, then no additional restrictions are applied for authorizing user access and the interface is open to everybody authenticated.

**Default:** undefined

This option in **multivalued**.

**Example:**
denyaccess

Synopsis: denyaccess = authgroup

Description: Defines that the specified authgroup members are REJECTED, not authorized to access the ARC-CE via this interface. Note that a related config option the allowaccess (see above) can be used to grant access. Multiple denyaccess and allowaccess authorization statements are allowed within a configuration block. These statements are processed sequentially in the order they are specified in the config block. The processing stops on first allowaccess or denyaccess statement matching the authgroup membership. If there are no authorization statements specified, then no additional restrictions are applied for authorizing user access and the interface is open to everybody authenticated.

Default: undefined

This option in multivalued.

Example:

denyaccess=blacklisted-users

maxjobdesc

Synopsis: maxjobdesc = size

Description: specifies maximal allowed size of job description in bytes. Default value is 5MB. Use 0 to set unlimited size.

Default: 5242880

Example:

maxjobdesc=0

[gridftp/filedir] block

The filedirplugin module of the gridftp server can be used to set up a simplistic grid storage element (SE). This subblock enables and configures such an SE by exporting a directory using the gridftp’s filedirplugin. Comment out this block if you don’t need a SE.

path

Synopsis: *path = virtdir

Description: The name of the virtual directory served by the gridftp server. The exported storage area is accessible as gsiftp://my_server/virtdir. Even / is a valid choice.

Default: undefined

Example:

path=/topdir
mount

Synopsis:  *mount = path

*Description:* The physical directory corresponding to the virtual one: gsiftp://my_server/virtdir will give access to this location.

*Default:* undefined

*Example:*

```
mount=/scratch/grid
```

allowaccess

*Synopsis: allowaccess = authgroup*

*Description:* (previously groupcfg) Defines that the specified authgroup members are authorized to access the gridftp file service. A related config option the denyaccess (see below) can be used to reject access. Multiple allowaccess and denyaccess authorization statements are allowed within a configuration block. These statements are processed sequentially in the order they are specified in the config block. The processing stops on first allowaccess or denyaccess statement matching the authgroup membership. If there are no authorization statements specified, then no additional restrictions are applied for authorizing user access and the interface is open to everybody authenticated.

*Default:* undefined

This option in *multivalued.*

*Example:*

```
allowaccess=biusers
denyaccess-atlasusers
```

denyaccess

*Synopsis: denyaccess = authgroup*

*Description:* Defines that the specified authgroup members are REJECTED, not authorized to access the gridftp file service. Note that a related config option the allowaccess (see above) can be used to grant access. Multiple denyaccess and allowaccess authorization statements are allowed within a configuration block. These statements are processed sequentially in the order they are specified in the config block. The processing stops on first allowaccess or denyaccess statement matching the authgroup membership. If there are no authorization statements specified, then no additional restrictions are applied for authorizing user access and the interface is open to everybody authenticated.

*Default:* undefined

This option in *multivalued.*

*Example:*

```
denyaccess-blacklisted-users
```

dir

*Synopsis: dir = path options*

*Description:* Specifies access rules for accessing files in *path* (relative to virtual and real path) and all the files and directories below. Available permissions check *options* are:
**nouser** do not use local file system rights, only use those specifies in this line

**owner** check only file owner access rights

**group** check only group access rights

**other** check only others access rights

If none of the above specified usual unix access rights are applied. Available permissions enforcement options are:

**read** allow reading files

**delete** allow deleting files

**append** allow appending files (does not allow creation)

**overwrite** allow overwriting already existing files (does not allow creation, file attributes are not changed)

**dirlist** allow obtaining list of the files

**cd** allow to make this directory current

**create owner:group permissions_or:permissions_and** allow creating new files. File will be owned by **owner** and owning group will be **group**. If * is used, the user/group to which connected user is mapped will be used. The permissions will be set to permissions_or & permissions_and. (second number is reserved for the future usage).

**mkdir owner:group permissions_or:permissions_and** allow creating new directories.

Example shows setting permissions on mounted / directory and adjusting permissions on /section1 and /section2 subdirectories.

This option in **multivalued**.

**Default:** undefined

**Example:**

```
dir=/ nouser read cd dirlist delete create *:* 664:664 mkdir *:* 775:775
dir=/section1 nouser read mkdir *:* 700:700 cd dirlist
dir=/section2 nouser read mkdir *:* 700:700 cd dirlist
```

**[infosys] block**

This block enables and configures the core part of the information system. Enables the information collection to be used by other ARC components, including interfaces. Parameters in this block applies to all the infosys subsystems.

**logfile**

**Synopsis:** logfile = path

**Description:** (previously providerlog) Specifies log file location for the information provider scripts.

**Default:** /var/log/arc/infoprovider.log

**Example:**

```
logfile=/var/log/arc/infoprovider.log
```

**Warning:** CHANGE: renamed
loglevel

Synopsis: loglevel = number

Description: (previously provider_loglevel) The loglevel for the infoprovider scripts (0-5). The infosys default is 1 (critical errors are logged) Each value corresponds to the following verbosity levels: FATAL => 0, ERROR => 1, WARNING => 2, INFO => 3, VERBOSE => 4, DEBUG => 5

Allowed values: 0, 1, 2, 3, 4, 5, FATAL, ERROR, WARNING, INFO, VERBOSE, DEBUG

Default: 1

Example:

```
loglevel=3
```

Warning: CHANGE: renamed

validity_ttl

Synopsis: validity_ttl = seconds

Description: The published infosys records advertise their validity e.g. how long the info should be considered up-to-date by the clients. Use this parameter to set the published validity value.

Default: 10800

Example:

```
validity_ttl=10800
```

[infosys/ldap] block

This infosys subblock enables and configures the ldap hosting service for the infosys functionality. Using an LDAP server with some schema is one way to publish information about your Computing Element. Comment out this block if you don’t want to run an LDAP-based information system.

Warning: CHANGE: new block

hostname

Synopsis: hostname = FQDN

Description: the hostname of the machine running the slapd service will be the bind for slapd. If not present, will be taken from the [common]

Default: $VAR{[common]hostname}

Example:

```
hostname=my.testbox
```
slapd_hostnamebind

Synopsis: `slapd_hostnamebind = string`

Description: 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 parameter is already sufficient. The example below will bind the slapd process to all the network interfaces available on the server.

Default: undefined

Example:

```
slapd_hostnamebind=*
```

port

Synopsis: `port = port_number`

Description: The port on which the slapd service runs. The default infosys port is assumed to be 2135 by many clients, therefore think twice before you change it because 3rd party clients assume 2135 to be the ldap infosys port.

Default: 2135

Example:

```
port=2135
```

user

Synopsis: `user = unix_user`

Description: overwrites the unix user running the slapd. By default the startup scripts search for well-known ldap-users like `ldap` or `openldap` than fall-back to `root` if not found.

Default: undefined

Example:

```
user=slapd
```

**Warning:** CHANGE: moved here from `[infosys]`

slapd

Synopsis: `slapd = path`

Description: explicitly define the path to slapd command. By default the startup scripts search for `slapd` binary in the system PATH.

Default: undefined

Example:

```
slapd=/usr/sbin/slapd
```
slapd_loglevel

**Synopsis:** slapd_loglevel = number

**Description:** Sets the native slapd loglevel (see man slapd). Slapd logs via syslog. 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.

**Default:** 0

**Example:**

```
slapd_loglevel=0
```

threads

**Synopsis:** threads = number

**Description:** The native slapd threads parameter, default is 32.

**Default:** 32

**Example:**

```
threads=128
```

timelimit

**Synopsis:** timelimit = seconds

**Description:** The native slapd timelimit parameter. Maximum number of seconds the slapd server will spend answering a search request. Default is 3600. You probably want a much lower value.

**Default:** 3600

**Example:**

```
timelimit=1800
```

idletimeout

**Synopsis:** idletimeout = seconds

**Description:** The native slapd idletimeout parameter. Maximum number of seconds the slapd server will wait before forcibly closing idle client connections. It’s value must be larger than the value of timelimit option. If not set, it defaults to timelimit + 1.

**Default:** $EVAL{$VAR{timelimit} + 1}

**Example:**

```
idletimeout=1801
```

infosys_ldap_run_dir

**Synopsis:** infosys_ldap_run_dir = path

**Description:** The location where NorduGrid/GLUE2 LDAP ldif file will be generated, and where the fifo to sync between infoprovers and BDII will be generated.
Default: /var/run/arc/infosys

Example:

```
infosys_ldap_run_dir=/var/run/arc/infosys
```

**ldap_schema_dir**

**Synopsis:** ldap_schema_dir = path

**Description:** Allows to explicitly specify an additional path to the schema files. Note that this doesn’t override standard location, but adds the specified path to the standard locations /etc/ldap and /etc/openldap. Normally it is sufficient to use only standard schema file locations, therefore not to set this parameter.

**Default:** undefined

**Example:**

```
ldap_schema_dir=/nfs/ldap/schema/
```

The following options configure the third-party bdii ldap parameters. In 99% of cases no need to change anything and use the defaults. These variables are usually automatically set by ARC, and are here mostly for debug purposes and to tweak exotic BDII installations.

**bdii_debug_level**

**Synopsis:** bdii_debug_level = level

**Description:** Set this parameter to DEBUG to check bdii errors in bdii-update.log At the same time don’t enable slapd logs this way reducing performance issues.

**Default:** WARNING

**Example:**

```
bdii_debug_level=ERROR
```

**bdii_provider_timeout**

**Synopsis:** bdii_provider_timeout = seconds

**Description:** (previously provider_timeout in bdii block) This variable allows a system administrator to modify the behaviour of bdii-update. This is the time BDII waits for the bdii provider scripts generated by A-REX infosys to produce their output.

**Default:** 10800

**Example:**

```
bdii_provider_timeout=10800
```

**Warning:** CHANGE: renamed

BDII5 uses these variables. These might change depending on BDII version. ARC sets them by inspecting distributed bdii configuration files. DO NOT
bdii_location

**Synopsis:** bdii_location = path  
**Description:** The installation directory for the BDII.  
**Default:** /usr  
**Example:**
```
bdii_location=/usr
```

bdii_run_dir

**Synopsis:** bdii_run_dir = path  
**Description:** Contains BDII pid files and slapd pid files  
**Default:** /var/run/arc/bdii  
**Example:**
```
bdii_run_dir=/var/run/arc/bdii
```

bdii_log_dir

**Synopsis:** bdii_log_dir = path  
**Description:** Contains infosys logs  
**Default:** /var/log/arc/bdii  
**Example:**
```
bdii_log_dir=/var/log/arc/bdii
```

bdii_tmp_dir

**Synopsis:** bdii_tmp_dir = path  
**Description:** Contains provider scripts  
**Default:** /var/tmp/arc/bdii  
**Example:**
```
bdii_tmp_dir=/var/tmp/arc/bdii
```

bdii_var_dir

**Synopsis:** bdii_var_dir = path  
**Description:** Contains slapd databases  
**Default:** /var/lib/arc/bdii  
**Example:**
```
```
bdii_var_dir=/var/lib/arc/bdii

**bdii_update_pid_file**

*Synopsis:* bdii_update_pid_file = path  
*Description:* Allows to change bdii-update pidfiles filename and location  
*Default:* $VAR{bdii_run_dir}/bdii-update.pid  
*Example:*  

```
bdii_update_pid_file=/var/run/arc/bdii/bdii-update.pid
```

**bdii_database**

*Synopsis:* bdii_database = backend_type  
*Description:* Configure what ldap database backend should be used.  
*Default:* hdb  
*Example:*  

```
bdii_database=hdb
```

**bdii_conf**

*Synopsis:* bdii_conf = path  
*Description:* Location of the bdii config file generated by ARC.  
*Default:* $VAR{[infosys/ldap]infosys_ldap_run_dir}/bdii.conf  
*Example:*  

```
bdii_conf=/var/run/arc/infosys/bdii.conf
```

**bdii_update_cmd**

*Synopsis:* bdii_update_cmd = path  
*Description:* path to bdii-update script  
*Default:* $VAR{bdii_location}/sbin/bdii-update  
*Example:*  

```
bdii_update_cmd=/usr/sbin/bdii-update
```

**bdii_db_config**

*Synopsis:* bdii_db_config = path  
*Description:* path to slapd database configuration file  
*Default:* /etc/arc/DB_CONFIG  
*Example:*
bdii_db_config=/etc/bdii/DB_CONFIG

**bdii_archive_size**

*Synopsis:* bdii_archive_size = number

*Description:* Sets BDII_ARCHIVE_SIZE in bdii configuration file

*Default:* 0

*Example:*

```
bdii_archive_size=0
```

**bdii_breathe_time**

*Synopsis:* bdii_breathe_time = number

*Description:* Sets BDII_BREATHE_TIME in bdii configuration file

*Default:* 10

*Example:*

```
bdii_breathe_time=10
```

**bdii_delete_delay**

*Synopsis:* bdii_delete_delay = number

*Description:* Sets BDII_DELETE_DELAY in bdii configuration file

*Default:* 0

*Example:*

```
bdii_delete_delay=0
```

**bdii_read_timeout**

*Synopsis:* bdii_read_timeout = number

*Description:* Sets BDII_READ_TIMEOUT in bdii configuration file

*Default:* $EVAL{$VAR{bdii_provider_timeout} + $VAR{[arex]infoproviders_timelimit} + $VAR{[arex]wakeupperiod}}

*Example:*

```
bdii_read_timeout=300
```

Infosys Schema sub-blocks: The following infosys sub-blocks enable information publishing according to various information schema. In order to publish information in a certain schema, the corresponding sub-block must be defined in addition to the schema-neutral [infosys/cluster] and [queue:name] blocks! Comment out a specific schema block if you don’t want to publish a specific information schema representation. Currently available information model (schema) sub-blocks:
[infosys/nordugrid] block

Enables the publication of the NorduGrid information model in the LDAP-based infosys. See the NORDUGRID-TECH-4 for schema definition. The configuration block does not contain any parameter. The information tree is populated based on the contents of the schema-neutral [infosys/cluster] and [queue:name] blocks.

**Warning:** CHANGE: new schema block. should be used to turn on/off nordugrid-ldap publication.

[infosys/glue2] block

Enables the publication of the GLUE2 information model both in the LDAP and XML rendering. The information tree is populated based on the contents of the schema-neutral [infosys/cluster] and [queue:name] blocks and the GLUE2 specific schema sub-blocks.

**Warning:** CHANGE: new schema block. should be used to turn on/off GLUE2 publication (both ldap & xml)

AdminDomain entity parameters: admindomain_name = string - The Name attribute for the admindomain. This will show in top-BDII to group the resources belonging to this cluster. To group a bunch of clusters under the same AdminDomain, just use the same name. If not specified, will default to UNDEFINEDVALUE.

**Default:** UNDEFINEDVALUE

**Example:**

```
admindomain_name=ARC-TESTDOMAIN
```

**Synopsis:** admindomain_description = text

**Description:** The free-form description of this domain.

**Default:** undefined

**Example:**

```
admindomain_description=ARC test Domain
```

**Synopsis:** admindomain_www = url

**Description:** The URL pointing at a site holding information about the AdminDomain.

**Default:** undefined

**Example:**
admindomain_www=http://www.nordugrid.org/

admindomain_distributed

Synopsis: admindomain_distributed = yes/no

Description: Set this to yes if the domain is distributed that means, if the resources belonging to the domain are considered geographically distributed.

Allowed values: yes, no

Default: no

Example:

admindomain_distributed=yes

admindomain_owner

Synopsis: admindomain_owner = email

Description: The contact email of a responsible person for the domain

Default: undefined

Example:

admindomain_owner=admin@nordugrid.org

admindomain_otherinfo

Synopsis: admindomain_otherinfo = text

Description: Free-form text that fills the OtherInfo GLUE2 field. no need to set, used only for future development.

Default: undefined

Example:

admindomain_otherinfo=Test Other info

ComputingService entity parameters: computingservice_qualitylevel = qlevel - (previously infosys_glue2_service_qualitylevel) Allows a sysadmin to define different GLUE2 QualityLevel values for A-REX. Refer to GLUE2 documentation for the qualitylevel definitions.

Allowed values: production, pre-production, testing, development

Default: production

Example:

computingservice_qualitylevel=production

Warning: CHANGE: renamed

end of the [infosys/glue2] schema block

Chapter 3. Documentation for Infrastructure Admins
[infosys/glue2/ldap] block

Enables the publication of the LDAP-rendering of the GLUE2 infomodel. The information tree is populated based on the contents of the schema-neutral [infosys/cluster] and [queue:name] blocks and the GLUE2 specific schema sub-blocks.

**Warning:** CHANGE: new schema block. should be used to turn on/off GLUE2 LDAP tree publication.

showactivities

*Synopsis:* showactivities = yes/no

*Description:* (previously infosys_glue2_ldap_showactivities) Enables GLUE2 ComputingActivities to appear in the LDAP rendering

*Allowed values:* yes, no

*Default:* no

*Example:*

```
showactivities=no
```

**Warning:** CHANGE: renamed.

[infosys/glue1] block

This block enables the publication of GLUE1 LDAP representation of a CE. The information tree is populated based on the contents of the schema-neutral [infosys/cluster] and [queue:name] blocks and the GLUE1 specific schema sub-blocks. This block holds information that is needed by the glue1, in addition to the schema-neutral blocks.

**Warning:** CHANGE: renamed. should be used to turn on/off GLUE1 publication.

resource_location

*Synopsis:* resource_location = string

*Description:* GLUE1 location attribute. IMPORTANT: no slashes or backslashes here!

*Default:* undefined

*Example:*

```
resource_location=Kastrup, Denmark
```

resource_latitude

*Synopsis:* resource_latitude = latitude

*Description:* GLUE1 latitude.

*Default:* undefined

*Example:*

```
resoue_latitude=55.75000

**resource_longitude**

*Synopsis:* resource_longitude = longitude  
*Description:* GLUE1 longitude.  
*Default:* undefined  
*Example:*  
resource_longitude=12.41670

**cpu_scaling_reference_s100**

*Synopsis:* cpu_scaling_reference_s100 = number  
*Description:* GLUE1 CPU_scaling  
*Default:* undefined  
*Example:*  
cpu_scaling_reference_s100=2400

**processor_other_description**

*Synopsis:* processor_other_description = string  
*Description:* GLUE1 proc description  
*Default:* undefined  
*Example:*  
processor_other_description=Cores 3, Benchmark 9.8-HEP-SPEC06

**glue_site_web**

*Synopsis:* glue_site_web = url  
*Description:* GLUE1 site web url  
*Default:* undefined  
*Example:*  
glue_site_web=http://www.ndgf.org

**glue_site_unique_id**

*Synopsis:* glue_site_unique_id = siteid  
*Description:* GLUE1 site id  
*Default:* undefined  
*Example:*  

[infosys/glue1/site-bdii] block

Enable this block ONLY if you want to publish a semi-fake GLUE1 site-bdii as part of the LDAP server. This block is used to configure ARC to generate a semi-fake site-bdii that can be registered in GOCDB.

**Warning:** CHANGE: renamed. turn on/off the fake site-bdii based on this block.

**unique_id**

*Synopsis:*  
`unique_id = id`

*Description:* The unique id used to identify this site, eg NDGF-T1

*Default:* undefined

*Example:*

`unique_id=NDGF-T1`

**url**

*Synopsis:*  
`url = url`

*Description:* The url of the resource BDII underlying the fake site bdii. It is on the format: ldap://host.domain:2170/mds-vo-name=something,o=grid.

*Default:* undefined

*Example:*

`url=ldap://host.domain:2170/mds-vo-name=something,o=grid`

[infosys/cluster] block

Information schema-neutral blocks [infosys/cluster] and [queue:NAME] contain attributes that describe the computing cluster together with its queues. The parameters are available for every information model/schema representation.

This block describes the cluster characteristics of a Computing Element. The information specified here is mostly used by the Infosys ARC component.

**alias**

*Synopsis:* alias = text

*Description:* An arbitrary alias name of the cluster, optional.

*Default:* undefined

*Example:*

`alias=Big Blue Cluster in Nowhere`
hostname

Synopsis: hostname = fqdn

Description: Set the FQDN of the frontend.

Default: $VAR{[common]hostname}

Example:
hostname=myhost.org

interactive_contactstring

Synopsis: interactive_contactstring = url

Description: the contact URL for interactive logins, set this if the cluster supports some sort of grid-enabled interactive login (gsi-ssh).

This option in multivalued.

Default: undefined

Example:
interactive_contactstring=gsissh://frontend.cluster:2200

collection

Synopsis: comment = text

Description: Free text field for additional comments on the cluster in a single line, no newline character is allowed!

Default: undefined

Example:
comment=This cluster is specially designed for XYZ applications: www.xyz.org

cluster_location

Synopsis: cluster_location = formatted_string

Description: The geographical location of the cluster, preferably specified as a postal code with a two letter country prefix

Default: undefined

Example:
cluster_location=DK-2100

cluster_owner

Synopsis: cluster_owner = text

Description: It can be used to indicate the owner of a resource, multiple entries can be used

This option in multivalued.

Default: undefined
Example:

<table>
<thead>
<tr>
<th>cluster_owner=World Grid Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster_owner=University of NeverLand</td>
</tr>
</tbody>
</table>

**advertisedvo**

*Synopsis*: advertisedvo = vo_name  
*Description*: (previously authorizedvo) This attribute is used to advertise which VOs are authorized on the cluster. Add only one VO for each advertisedvo entry. Multiple VOs in the same line will cause errors. These entries will be shown in all GLUE2 AccessPolicy and MappingPolicy objects, that is, they will apply for all Endpoints(Interfaces) and all Shares(currently queues). You can also add additional advertisedvos to queues. The information is also published in the NorduGrid schema.

**Note:**

It is IMPORTANT to understand that this parameter is NOT enforcing any access control, it is just for information publishing!

This option in **multivalued**.

**Default**: undefined

**Example**:

<table>
<thead>
<tr>
<th>advertisedvo-atlas</th>
</tr>
</thead>
<tbody>
<tr>
<td>advertisedvo-community.nordugrid.org</td>
</tr>
</tbody>
</table>

**Warning**: CHANGE: renamed it as advertisedvo

**clustersupport**

*Synopsis*: clustersupport = email  
*Description*: This is the support email address of the resource.

This option in **multivalued**.

**Default**: undefined

**Example**:

| clustersupport-arc.support@mysite.org |
| clustersupport-arc.support@myproject.org |

**homogeneity**

*Synopsis*: homogeneity = True/False  
*Description*: 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. False may trigger multiple GLUE2 ExecutionEnvironments to be published if applicable.

**Allowed values**: True, False

**Default**: True
architecture

Synopsis: architecture = string

Description: 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.

Default: adotf

Example:

```
architecture=adotf
```

opsys

Synopsis: opsys = formatted_string

Description: 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. If opsys=adotf, will result in Automatic Determination of the Operating System On The Frontend, which should only be used if the frontend has the same OS as the nodes. The adotf discovered values will be used to fill GLUE2 OSName, OSVersion and OSFamily unless these values are explicitly defined for each queue. See the `[queue:queueName]` block for their usage. Note: Any custom value other than ‘adotf’ does NOT affect values in the GLUE2 schema.

This option in multivalued.

Default: adotf

Example:

```
opsys=Linux-2.6.18
opsys=glibc-2.5.58
opsys=CentOS-5.6
```

nodecpu

Synopsis: nodecpu = formatted_string

Description: This is the cputype of the homogeneous nodes. The string is constructed from the `/proc/cpuinfo` as the value of `model name` 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.

Default: adotf

Example:
nodememory

**Synopsis:** nodememory = number

**Description:** 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.

**Default:** undefined

**Example:**

nodememory=64000

benchmark

**Synopsis:** benchmark = name value

**Description:** This optional multivalued attribute can be used to specify benchmark results on the cluster level. Use this cluster attribute if only the NODES are homogeneous with respect to the benchmark performance. Otherwise the similar queue-level attribute should be used. Please try to use standard benchmark names, if possible.

This option in **multivalued**.

**Default:** undefined

**Example:**

benchmark=SPECINT2000 222
benchmark=SPECFP2000 333

middleware

**Synopsis:** middleware = string

**Description:** The multivalued attribute shows the installed grid software on the cluster. Nordugrid-ARC is automatically set, no need to specify

This option in **multivalued**.

**Default:** undefined

**Example:**

middleware=my software

nodeaccess

**Synopsis:** nodeaccess = inbound/outbound

**Description:** 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.

This option in **multivalued**.
Default: undefined

Allowed values: inbound, outbound

Example:

<table>
<thead>
<tr>
<th>nodeaccess=inbound</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeaccess=outbound</td>
</tr>
</tbody>
</table>

**localse**

Synopsis: localse = url

Description: This multivalued parameter tells the BROKER that certain URLs (and locations below that) should be considered locally available to the cluster.

This option in multivalued.

Default: undefined

Example:

<table>
<thead>
<tr>
<th>localse=gsiftp://my.storage/data1/</th>
</tr>
</thead>
<tbody>
<tr>
<td>localse=gsiftp://my.storage/data2/</td>
</tr>
</tbody>
</table>

**cpudistribution**

Synopsis: cpudistribution = formatted_string

Description: This is the CPU distribution over nodes given in the form \( ncpu:m \) where:

- \( n \) is the number of CPUs per machine
- \( m \) is the number of such machines

Example: 1cpu:3,2cpu:4,4cpu:1 represents a cluster with 3 single CPU machines, 4 dual CPU machines and one machine with 4 CPUs.

Default: undefined

Example:

| cpudistribution=1cpu:3,2cpu:4,4cpu:1 |

**maxcputime**

Synopsis: maxcputime = number

Description: This is the maximum CPU time specified in seconds that the LRMS can allocate for the job. The default if not defined is that infoprotocols get this value automatically from the LRMS. The purpose of this option is to tweak and override discovered value, or publish this value in case the LRMS module do not support automatic detection.

Default: undefined

Example:

| maxcputime=300000 |
**maxwalltime**

*Synopsis:* maxwalltime = number

*Description:* This is the maximum Wall time specified in seconds that the LRMS can allocate for the job. The default if not defined is that infoproviders get this value automatically from the LRMS. The purpose of this option is to tweak and override discovered value, or publish this value in case the LRMS module do not support automatic detection.

*Default:* undefined

*Example:*

maxwalltime=600000

---

**[queue:name] block**

Each grid-enabled queue on the cluster should be represented and described by a separate queue block. The queue_name should be used as a label in the block name. In case of fork, or other LRMSes with no queue names, just use any unique string. A queue can represent a PBS/LSF/SGE/SLURM/LL queue, a SGE pool, a Condor pool or a single machine in case ‘fork’ type of LRMS. This block describes the queue characteristics.

---

**homogeneity**

*Synopsis:* homogeneity = True/False

*Description:* 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. Possible values: True,False, the default is True.

*Allowed values:* True, False

*Default:* $VAR{[infosys/cluster]homogeneity}

*Example:*

homogeneity=True

---

**comment**

*Synopsis:* comment = text

*Description:* A free-form text field for additional comments on the queue in a single line, no newline character is allowed!

*Default:* undefined

*Example:*

comment=This queue is nothing more than a condor pool

---

**pbs_queue_node**

*Synopsis:* pbs_queue_node = string

*Description:* (previously queue_node_string) In PBS you can assign nodes to a queue (or a queue to nodes) by using the node property mark in PBS config.

Essentially, pbs_queue_node value is used to construct nodes= string in PBS script, such as nodes=count:pbs_queue_node where count is taken from the job description (1 if not specified).
This corresponds to setting the following parameter in PBS for this queue:

```
resources_default.neednodes = cpu_topology[:pbs_queue_node]
```

Setting the \texttt{pbs_queue_node} changes how the queue-totalcpus, user freecpus are determined for this queue. You shouldn’t use this option unless you are sure that your PBS configuration makes use of the above configuration. Read NorduGrid PBS instructions for more information: http://www.nordugrid.org/documents/pbs-config.html

\textit{Default:} undefined

\textit{Example:}

```
pbs_queue_node=gridlong_nodes
pbs_queue_node=ppn=4:ib
```

\textbf{Warning:} CHANGE: renamed

\textbf{sge\_jobopts}

\textit{Synopsis:} \texttt{sge\_jobopts = string}

\textit{Description:} Per-queue override of additional SGE options to be used when submitting jobs to SGE to this queue

\textit{Default:} undefined

\textit{Example:}

```
sge\_jobopts=-P atlas -r yes
```

\textbf{condor\_requirements}

\textit{Synopsis:} \texttt{condor\_requirements = string}

\textit{Description:} It may be defined for each Condor queue. Use this option to determine which nodes belong to the current queue. The value of \texttt{condor\_requirements} must be a valid constraints string which is recognized by a \texttt{condor\_status -constraint ...} command. It can reference pre-defined ClassAd attributes (like Memory, OpSys, Arch, HasJava, etc) but also custom ClassAd attributes. To define a custom attribute on a condor node, just add two lines like the ones below in the \texttt{$(hostname).local} config file on the node:

```
NORDUGRID\_RESOURCE=TRUE
STARTD\_EXPRS = NORDUGRID\_RESOURCE, $(STARTD\_EXPRS)
```

A job submitted to this queue is allowed to run on any node which satisfies the \texttt{condor\_requirements} constraint. If \texttt{condor\_requirements} is not set, jobs will be allowed to run on any of the nodes in the pool. When configuring multiple queues, you can differentiate them based on memory size or disk space, for example.

\textit{Default:} $VAR([lrms]\texttt{condor\_requirements})

\textit{Example:}

```
condor\_requirements=(OpSys == "linux" && NORDUGRID\_RESOURCE && Memory >= 1000 && Memory < 2000)
```

\textbf{slurm\_requirements}

\textit{Synopsis:} \texttt{slurm\_requirements = string}
**Description:** Use this option to specify extra SLURM-specific parameters.

**Default:** undefined

**Example:**

```
slurm_requirements-memory on node >> 200
```

**Warning:** CHANGE: new parameter proposed in Umea retreat

### totalcpus

**Synopsis:** totalcpus = number  

**Description:** Manually sets the number of cpus assigned to the queue. No need to specify the parameter in case the queue_node_string method was used to assign nodes to the queue (this case it is dynamically calculated and the static value is overwritten) or when the queue have access to the entire cluster (this case the cluster level totalcpus is the relevant parameter).

**Default:** undefined

**Example:**

```
totalcpus=32
```

Queue-level configuration parameters: nodecpu, nodememory, architecture, opsys and benchmark should be set if they are homogeneous over the nodes assigned to the queue AND they are different from the cluster-level value. Their meanings are described in the [infosys/cluster] block. Usage: this queue collects nodes with nodememory=512 while another queue has nodes with nodememory=256 -> don’t set the cluster attributes but use the queue-level attributes. When the frontend’s architecture or cputype agrees with the queue nodes, the adotf (Automatically Determine On The Frontend) can be used to request automatic determination of architecture or nodecpu. For GLUE2, fine tune configuration of ExecutionEnvironments' OSName, OSVersion, OSFamily is allowed with dedicated options osname,osversion,osfamily.

### nodecpu

**Synopsis:** nodecpu = formatted_string  

**Description:** see description at [infosys/cluster] block

**Default:** $VAR{[infosys/cluster]nodecpu}

**Example:**

```
nodecpu=AMD Duron(tm) Processor @ 700 MHz
```

### nodememory

**Synopsis:** nodememory = number  

**Description:** see description at [infosys/cluster] block

**Default:** $VAR{[infosys/cluster]nodememory}

**Example:**

```
nodememory=512
```
defaultmemory

Synopsis: defaultmemory = number

Description: The LRMS memory request of job to be set by the LRMS backend scripts, if a user submits a job without specifying how much memory should be used. The order of precedence is: job description -> [lrms-defaultmemory] -> [queue-defaultmemory]. This is the amount of memory (specified in MB) that a job will request.

Default: undefined

Example:

defaultmemory=512

Warning: CHANGE: new parameter in this block

architecture

Synopsis: architecture = string

Description: see description at [infosys/cluster] block

Default: $VAR{[infosys/cluster]architecture}

Example:

architecture=adotf

opsys

Synopsis: opsys = formatted_string

Description: see description at [infosys/cluster] block If osname, osversion are present, the values in opsys are ignored.

This option in multivalued.

Default: $VAR{[infosys/cluster]opsys}

Example:

opsys=Linux-2.6.18
opsys=glibc-2.5.58

osname

Synopsis: osname = string

Description: Only for GLUE2 overrides values defined in opsys for a single ExecutionEnvironment. Configuration of multiple ExecutionEnvironment for the same queue is not supported. Create a different queue for that.

Default: undefined

Example:

osname=Ubuntu
NorduGrid ARC 6 Information, Release ARC6

osversion

Synopsis: osversion = string

Description: Only for GLUE2 overrides values defined in opsys for a single ExecutionEnvironment. Configuration of multiple ExecutionEnvironment for the same queue is not supported. Create a different queue for that.

Default: undefined

Example:

```
osversion=12.04
```

osfamily

Synopsis: osfamily = string

Description: Only for GLUE2 overrides values defined in opsys for a single ExecutionEnvironment. Configuration of multiple ExecutionEnvironment for the same queue is not supported. Create a different queue for that.

Default: undefined

Example:

```
osfamily=linux
```

benchmark

Synopsis: benchmark = name value

Description: see description at [infosys/cluster] block

This option in multivalued.

Default: $VAR{[infosys/cluster]benchmark}

Example:

```
benchmark=SPECINT2000 222
benchmark=SPECFP2000 333
```

allowaccess

Synopsis: allowaccess = authgroup

Description: (previously groupcfg) Defines that the specified authgroup members are authorized to submit jobs to this queue of ARC-CE after the user already granted access to the CE via one of the interfaces. A related config option the denyaccess (see below) can be used to deny submission to the queue. Multiple allowaccess and denyaccess authorization statements are allowed within a configuration block. These statements are processed sequentially in the order they are specified in the config block. The processing stops on first allowaccess or denyaccess statement matching the authgroup membership. If there are no authorization statements specified, then the queue is accessible by everyone already authorized.

Default: undefined

This option in multivalued.

Example:

```
allowaccess=biosusers
allowaccess=atlasusers
```
denyaccess

Synopsis: denyaccess = authgroup

Description: Defines that the specified authgroup members are NOT allowed to submit jobs to this queue of ARC-CE after despite the user is already granted access to the CE via one of the interfaces. A related config option the allowaccess (see below) can be used to grant job submission to the queue. Multiple allowaccess and denyaccess authorization statements are allowed within a configuration block. These statements are processed sequentially in the order they are specified in the config block. The processing stops on first allowaccess or denyaccess statement matching the authgroup membership. If there are no authorization statements specified, then the queue is accessible by everyone already authorized.

Default: undefined

This option in multivalued.

Example:

denyaccess=blacklisted-for-the-queue

advertisedvo

Synopsis: advertisedvo = vo_name

Description: (previously authorizedvo) This attribute is used to advertise which VOs are authorized on the [queue:name] of the cluster. Add only one VO for each advertisedvo entry. Multiple VOs in the same line will cause errors. These entries will be shown in the MappingPolicy objects, that is, they will apply for the Shares that corresponds to the queue. The information is also published in the NorduGrid schema.

Note: if you have also configured advertisedvo in the [infosys/cluster] block, the result advertised VOs per queue is the union set of what is contained in the [infosys/cluster] and in this [queue:name] block!

Note: it is IMPORTANT to understand that this parameter is NOT enforcing any access control, it is just for information publishing!

This option in multivalued.

Default: undefined

Example:

advertisedvo-atlas
advertisedvo-community.nordugrid.org

Warning: CHANGE: renamed it as advertisedvo

maxslotsperjob

Synopsis: maxslotsperjob = number

Description: This GLUE2 specific parameter configures the MaxSlotsPerJob value on a particular queue. This value is usually generated by LRMS infocollectors, but there are cases in which a system administrator might like to tweak it. Default is to publish what is returned by the LRMS, and if nothing is returned, NOT to publish the MaxSlotsPerJob attribute. If a system administrator sets the value here, that value will be published instead, regardless of what the LRMS returns. Each LRMS might have a different meaning for this value.
Default: undefined

Example:

maxslotsperjob=5

forcedefaultvoms

Synopsis: forcedefaultvoms = VOMS_FQAN

Description: specify VOMS FQAN which user will be assigned if his/her credentials contain no VOMS attributes.

Default: $VAR{[arex]forcedefaultvoms}

Example:

forcedefaultvoms=/vo/group/subgroup

Warning: CHANGE: documented in [queue] as described in [arex]

maxcputime = number - This value overrides the one defined in the [infosys/cluster] block. See description in that block.

Default: undefined

Example:

maxcputime=300000

maxwalltime

Synopsis: maxwalltime = number

Description: This value overrides the one defined in the [infosys/cluster] block. See description in that block.

Default: undefined

Example:

maxwalltime=600000

[datadelivery-service] block

This block configures and enables the data delivery service. This service is intended to off-load data-staging from A-REX and usually deployed on one or more separate machines.

This service can also act as an independent data transfers service that case it would require an inteligent data manager that could replace A-REX’s intelligence.

Warning: CHANGE: NEW block

transfer_dir

Synopsis: *transfer_dir = path

Description: (previously allowed_dir) The directori(es) on the DDS host in which the service is allowed to read and write. When DDS is used as a remote transfer service assisting A-REX then this is usually one or more cache and/or session directories shared as a common mount with A-REX.
This option in **multivalued**.

**Default:** undefined

**Example:**

```
transfer_dir=/shared/arc/cache
transfer_dir=/shared/arc/session
```

**Warning:** CHANGE: NEW, also renamed from allowed_dir to work_dir

### hostname

**Synopsis:** hostname = FQDN

**Description:** The hostname of the machine on which DDS service runs.

**Default:** $EXEC{hostname -f}

**Example:**

```
hostname=localhost
```

**Warning:** CHANGE: NEW

### port

**Synopsis:** port = port

**Description:** Port on which service listens

**Default:** 443

**Example:**

```
port=8443
```

**Warning:** CHANGE: NEW

### pidfile

**Synopsis:** pidfile = path

**Description:** pid file of the daemon

**Default:** /var/run/arched-datadelivery-service.pid

**Example:**

```
pidfile=/var/run/arched-datadelivery-service.pid
```

**Warning:** CHANGE: NEW
logfile

Synopsis: logfile = path
Description: log file of the daemon
Default: /var/log/arc/datadelivery-service.log
Example:

logfile=/tmp/delivery.log

Warning: CHANGE: NEW

loglevel

Synopsis: loglevel = level
Description: set loglevel of the data delivery service between 0 (FATAL) and 5 (DEBUG). Defaults to 3 (INFO).
Allowed values: 0, 1, 2, 3, 4, 5
Default: 3
Example:

loglevel=4

Warning: CHANGE: NEW

user

Synopsis: user = username
Description: Overwrites the user under which the service runs. The default is the user starting the service. DDS is very limited if not run as root.
Default: undefined
Example:

user=ddsuser

Warning: CHANGE: NEW

secure

Synopsis: secure = yes/no
Description: Set to no if the service should run without a host certificate. In this case the corresponding deliveryservice option in the [arex/data-staging] A-REX configuration block should use http rather than https URLs.
Allowed values: yes, no
Default: yes
Example:

```
secure=no
```

**Warning:** CHANGE: NEW

### allowed_ip

**Synopsis:** `allowed_ip = ip`

**Description:** IP address authorized to access service. Normally this is the A-REX host IP. By default the delivery service listens on all available interfaces, so if both IPv4 and IPv6 are enabled on this and the A-REX host, remember to add both A-REX host IPs here.

This option in **multivalued**.

**Default:** undefined

**Example:**

```
allowed_ip=192.0.2.1
allowed_ip=2001:db8:85a3::8a2e:370:7334
```

**Warning:** CHANGE: NEW

### allowed_dn

**Synopsis:** `allowed_dn = DN`

**Description:** DN authorized to access service. This option restricts access to specified DNs (of the users who submit jobs to A-REX). It is only effective if `secure=yes`.

This option in **multivalued**.

**Default:** undefined

**Example:**

```
allowed_dn=/O=Grid/O=Big VO/CN=Main Boss
```

**Warning:** CHANGE: NEW

### x509_host_key

**Synopsis:** `x509_host_key = path`

**Description:** Optional parameter to overwrite `[common]` block values.

**Default:** `$VAR{[common]x509_host_key}`

**Example:**

```
x509_host_key=/etc/grid-security/hostkey.pem
```
**x509_host_cert**

**Synopsis:** x509_host_cert = path  
**Description:** Optional parameter to overwrite [common] block values.  
**Default:** $VAR{[common]x509_host_cert}  
**Example:**

```
x509_host_cert=/etc/grid-security/hostcert.pem
```

**Warning:** CHANGE: new in this block

**x509_cert_dir**

**Synopsis:** x509_cert_dir = path  
**Description:** Optional parameter to overwrite [common] block values.  
**Default:** $VAR{[common]x509_cert_dir}  
**Example:**

```
x509_cert_dir=/etc/grid-security/certificates
```

**Warning:** CHANGE: new in this block

**[acix-scanner] block**

The ARC Cache Index (ACIX) is a distributed system that maintains a catalog of locations of cached files stored in various A-REX caches. ACIX consists of two components, the Cache Scanner (on the CEs), and the Index Server.

This config block enables and configures the cache scanner component of ACIX. The scanning component of ACIX is a separate service that runs alongside A-REX/DDS and all it needs from A-REX/DDS service is the location of the cache.

The acix-scanner periodically scans the cache directories and composes a Bloom filter of A-REX cache content which can be pulled via its public interface. In the current deployment the ACIX index services are the main consumers of the collected information.

**Warning:** CHANGE: renamed block, renamed component (cache scanner, instead of cacheserver)

**cachedir**

**Synopsis:** cachedir = cache_path  
**Description:** Specifies the cache directory to be scanned in case not set in [arex/cache] block (e.g. the scanner is running on a different machine than A-REX)
This option is **multivalued**.

**Default:** $VAR{[arex/cache]cachedir}

**Example:**

| cachedir=/scratch/cache
| cachedir=/fs1/cache drain

### logfile

**Synopsis:** logfile = path

**Description:** Log file location for the acix-scanner.

**Default:** /var/log/arc/arc-acix-scanner.log

**Example:**

logfile=/tmp/arc-acix-scanner.log

**Warning:** CHANGE: modified default for the logfile

### hostname

**Synopsis:** hostname = string

**Description:** Hostname on which the acix-scanner listens

**Default:** $EXEC(hostname -f)

**Example:**

hostname=myhost.org

### port

**Synopsis:** port = port

**Description:** Port on which the acix-scanner service listens

**Default:** 5443

**Example:**

port=6000

### cachedump

**Synopsis:** cachedump = yes/no

**Description:** Whether to make a dump of the list of files on the cache at $TMP/ARC-ACIX/timestamp each time the acix-scanner runs.

**Default:** no

**Allowed values:** yes, no

**Example:**

...
cachedump=yes

**x509_host_key**

*Synopsis:* `x509_host_key = path`

*Description:* Optional parameter to overwrite [common] block values.

*Default:* $VAR{[common]x509_host_key}

*Example:*

```bash
x509_host_key=/etc/grid-security/hostkey.pem
```

**Warning:** CHANGE: new in this block

**x509_host_cert**

*Synopsis:* `x509_host_cert = path`

*Description:* Optional parameter to overwrite [common] block values.

*Default:* $VAR{[common]x509_host_cert}

*Example:*

```bash
x509_host_cert=/etc/grid-security/hostcert.pem
```

**Warning:** CHANGE: new in this block

**x509_cert_dir**

*Synopsis:* `x509_cert_dir = path`

*Description:* Optional parameter to overwrite [common] block values.

*Default:* $VAR{[common]x509_cert_dir}

*Example:*

```bash
x509_cert_dir=/etc/grid-security/certificates
```

**Warning:** CHANGE: new in this block

**[acix-index] block**

The ARC Cache Index (ACIX) is a distributed system that maintains a catalog of locations of cached files stored in various A-REX caches. ACIX consists of two components, the Cache Scanner (on the CEs), and the Index Server. This config block enables and configures the index server component of ACIX.

The index server component of ACIX collects cache content filters generated by a set of acix-scanners and maintains an aggregated view of distributed cache contents.
The acix-index server is deployed separately and can be queried for the location of cached files. The service endpoint is https://hostname:6443/data/index and a query is performed via giving the URLs to check as comma-separated values to the url option, e.g.: index_service_endpoint?url=http://www.nordugrid.org:80/data/echo.sh,http://my.host/data1

cachescanner

**Synopsis:** *cachescanner = url*

**Description:** (previously cacheserver) ACIX cache scanners from which to pull information

This option in **multivalued**.

**Default:** undefined

**Example:**

```
cachescanner=https://some.host:5443/data/cache
cachescanner=https://another.host:5443/data/cache
```

**Warning:** CHANGE: renamed

x509_host_key

**Synopsis:** x509_host_key = path

**Description:** Optional parameter to overwrite [common] block values.

**Default:** $VAR{[common]x509_host_key}

**Example:**

```
x509_host_key=/etc/grid-security/hostkey.pem
```

**Warning:** CHANGE: new in this block

x509_host_cert

**Synopsis:** x509_host_cert = path

**Description:** Optional parameter to overwrite [common] block values.

**Default:** $VAR{[common]x509_host_cert}

**Example:**

```
x509_host_cert=/etc/grid-security/hostcert.pem
```

**Warning:** CHANGE: new in this block

x509_cert_dir

**Synopsis:** x509_cert_dir = path

**Description:** Optional parameter to overwrite [common] block values.
Default: $VAR{[common]x509_cert_dir}

Example:

```
x509_cert_dir=/etc/grid-security/certificates
```

Warning: CHANGE: new in this block

[userlist:name] block

The [userlist:name] blocks are used to define userlists and configure how those are generated by the nordugridmap utility, including the optional user mapping information. The userlist is identified with the generated file that is stored in the outfile. Please note that behaviour of the nordugridmap external utility can be further modified by the optional [nordugridmap] block (see next block). Note that the [userlist:name] block by itself does not affect authorization. In order to define auth rules userlists can be referenced within the [authgroup] blocks by the userlist-name parameter. Also, the generated outfile can be used via the ‘file’ auth rule of the [authgroup] block. The order of this block in arc.conf may influence authorization decision. This block must appear before the [arex] block.

Warning: CHANGE: new blockname that better reflects the purpose of the config block. This block is not defining any VO!

outfile

Synopsis: outfile = path

Description: (previously file) The full path of the GENERATED file that contains the userlist (with optional mapping info). If the same file specified as output for different [userlist:name] blocks then nordugridmap will automatically merge entries following the order of the blocks.

Default: /etc/grid-security/grid-mapfile

Example:

```
outfile=/etc/grid-security/lists/atlas-users
```

Warning: CHANGE: renamed from file

source

Synopsis: *source = url

Description: the URL of the VO database which is used to generate the userlist. The nordugridmap will use this URL to automatically generate and keep up-to-date userlist (mapfile) specified by the ‘outfile’ attribute.

url is a multivalued attribute, several sources can be specified and all the users from those sources will be merged into the same file. The source URLs are processed in the given order.

Currently supported URL types are:

- **http(s)**: // URL to plain text file. File should contain a list of DNs.
- **voms(s)**: // URL to VOMS-Admin interface
**file://** local file (stand-alone or dynamically generated by nordugridmap). File should contain a list of DNs with optional mapped unixid: user DN [mapped user ID]. Result of optional mapped unixid processing depend on `mapuser_processing` option settings.

**userlist://** reference to another [userlist/name] configuration block

**nordugrid** add NorduGrid VO members

You can use either `userlist://` or `file://` entries to specify dependencies between [userlist/name] blocks, but using `userlist://` is a recommended way.

For each separate source URL it is possible to override some parameters value. You can use the following syntax to perform this:

"source=URL < parameter1=value1 parameter2=value2"

You can override the following parameters:

- **mapped_unixid** for http(s),voms(s),ldap and file URLs
- **cache_enable** for http(s),voms(s),ldap and file URLs
- **voms_method** for voms(s) URLs
- **mapuser_processing** for file URLs with `mapped_unixid='<unixid>'` override (control `mapped_unixid` overriding behaviour for URL)

This option in **multivalued**.

**Default:** undefined

**Example:**

```
source=vomss://voms.ndgf.org:8443/voms/nordugrid.org
source=vomss://lcg-voms.cern.ch:8443/voms/atlas?/atlas/Role=VO-Admin < mapped_unixid=atlasadmin
source=vomss://kuiken.nikhef.nl:8443/voms/gin.ggf.org < voms_method=get
source=http://www.nordugrid.org/developers.dn
source=file:///etc/grid-security/priviliged_users.dn
source=userlist://biousers
source=nordugrid
```

**Warning:** CHANGE: MODIFIED options: edg-mkgridmap source option is removed (obsolete technology); vo source option is renamed as userlist.

**mapped_unixid**

**Synopsis:** `mapped_unixid = unixid`

**Description:** The local UNIXID which is optionally used in the generated outfile by the nordugridmap utility.

If any of the sources have already provided mapping information (file:// or userlist://) behavior depends on ‘mapuser_processing’ from the [nordugridmap] block:

- **mapuser_processing = overwrite** ignore already provided mapping and apply `mapped_unixid` for all sources
- **mapuser_processing = keep** apply `mapped_unixid` only for sources that does not already have mapping information

If the `mapped_unixid` config parameter is not specified or has empty value, then behavior depends on the value of `allow_empty_unixid` from the [nordugridmap] block:
allow_empty_unixid = yes  empty value will be used for mapped_unixid which means
that nordugridmap will generate only the list of DNs without mapping (consider using
mapuser_processing = overwrite along with this option or sources that does not pro-
vide previously defined mapping information)

allow_empty_unixid = no then nordugridmap will skip users without mapping information
(if no mapping information provided by sources)

Default: nobody

Example:

| mapped_unixid= |
| mapped_unixid=gridtest |

Warning: CHANGE: Modified. The empty value should work the same way as missing parameter. set the
default to nobody

voms_fqan_map

Synopsis: voms_fqan_map = fqan unixid

Description: The local UNIXID which is used to map voms(s) sources with specific FQAN given.

Several voms_fqan_map can be specified for a [userlist/name] block. For each voms(s) sources in
[userlist/name] block and every voms_fqan_map record separate source record will be automatically gen-
erated with mapped_unixid overwritten to specified one.

Sources are generated in a given voms_fqan_map order. Original voms(s) source URL are processed LAST.
This allows to simplify configuration, especially in redundancy cases when several VOMS servers are used for the
same VO.

This option in multivalued.

Default: undefined

Example:

```
voms_fqan_map=/atlas/Role=VO-Admin atlasadmin
voms_fqan_map=/atlas/Role=production atlasprod
```
[nordugridmap] block

This optional block is used to fine-tune the behavior of the nordugridmap utility - an ARC tool used to generate grid-mapfiles. Normal setups don’t need to configure this block. Please refer to [userlist/name] block above to find information how to specify sources for userlist generation. This section setup general source-independent parameters.

x509_host_key

Synopsis: x509_host_key = path

Description: Optional parameter to overwrite [common] block values.

Default: $VAR{[common]x509_host_key}

Example:

```
x509_host_key=/etc/grid-security/hostkey.pem
```

Warning: CHANGE: renamed

x509_host_cert

Synopsis: x509_host_cert = path

Description: Optional parameter to overwrite [common] block values.

Default: $VAR{[common]x509_host_cert}

Example:

```
x509_host_cert=/etc/grid-security/hostcert.pem
```

Warning: CHANGE: renamed

x509_cert_dir

Synopsis: x509_cert_dir = path

Description: Optional parameter to overwrite [common] block values.

Default: $VAR{[common]x509_cert_dir}

Example:

```
x509_cert_dir=/etc/grid-security/certificates
```

gridmap_owner

Synopsis: gridmap_owner = username

Description: The owner of the generated mapfiles.

Default: root

Example:
gridmap_owner=root

gridmap_group

Synopsis: gridmap_group = groupname
Description: The group of generated gridmap files.
Default: root
Example:
gridmap_group=root

gridmap_permissions

Synopsis: gridmap_permissions = filemode
Description: The permissions of generated gridmap files.
Default: 0600
Example:
gridmap_permissions=0600

log_to_file

Synopsis: log_to_file = yes/no
Description: control whether logging output of nordugridmap will be saved to file. If the value is ‘no’ nordugridmap will write all information to STDERR.
Allowed values: yes, no
Default: yes
Example:
log_to_file=no

logfile

Synopsis: logfile = path
Description: specify the nordugridmap log file location when log_to_file is set yes.
Default: /var/log/arc/nordugridmap.log
Example:
logfile=/var/log/arc/nordugridmap.log
cache_enable

Synopsis: cache_enable = yes/no

Description: Controls whether caching of external sources will be used.

Allowed values: yes, no

Default: yes

Example:

```bash
cache_enable=yes
```

cachedir

Synopsis: cachedir = path

Description: Specifies the path where cached sources will be stored.

Default: /var/spool/arc/gridmapcache/

Example:

```bash
cachedir=/var/spool/arc/gridmapcache/
```

cachetime

Synopsis: cachetime = seconds

Description: Controls how long (in seconds) the cached information remains valid. Default is 259200 (3 days).

Default: 259200

Example:

```bash
cachetime=259200
```

mapuser_processing

Synopsis: mapuser_processing = overwrite/keep

Description: Controls the behavior of [userlist:name] block’s mapped_unixid parameter usage. Please see ‘mapped_unixid’ description in [userlist:name] block for details.

Allowed values: keep, overwrite

Default: keep

Example:

```bash
mapuser_processing=keep
```

allow_empty_unixid

Synopsis: allow_empty_unixid = yes/no

Description: Controls whether empty (or unspecified) ‘mapped_unixid’ [userlist:name] block option is allowed to be used. Please see ‘mapped_unixid’ description for details.

Allowed values: yes, no
Default: yes

Example:

```
allow_empty_unixid=no
```

**Warning:** CHANGE: MODIFIED. set default to yes.

---

**voms_method**

**Synopsis:** voms_method = soap/get

**Description:** Controls how to get information from VOMS(S) sources. Valid values are:

- soap - call SOAP method directly using SOAP::Lite
- get - use old implementation that manually parses XML response

**Allowed values:** soap, get

**Default:** soap

**Example:**

```
voms_method=soap
```

---

**loglevel**

**Synopsis:** loglevel = level

**Description:** (previously debug) Controls the verbosity of nordugridmap output. Valid values are:

- 0 - FATAL - only critical fatal error shown
- 1 - ERROR - errors, including non-critical are shown
- 2 - WARNING (default) - configuration errors that can be ignored
- 3 - INFO - processing information
- 4 - VERBOSE - a bit more processing information
- 5 - DEBUG - lot of processing information

When test run is requested (–test command line option of the nordugridmap) loglevel is automatically set to 5 (DEBUG).

**Allowed values:** 0, 1, 2, 3, 4, 5

**Default:** 2

**Example:**

```
loglevel=4
```

**Warning:** CHANGE: renamed as loglevel

---

**fetch_timeout**

**Synopsis:** fetch_timeout = seconds

**Description:** Controls how many time (in seconds) nordugridmap will wait for external sources retrieval.

**Default:** 15

**Example:**
3.2 ARC Computing Element Deployment and Operations

3.2.1 Try ARC6: towards distributed computing in a few minutes

Scared of distributed computing complexities?

With ARC6 you can setup a Computing Element and try common distributed computing workflows in just a few minutes!

ARC6 comes with so-called zero configuration included and works out of the box without any configuration at all.

Step 1. Enable NorduGrid ARC6 repos

Prepare your system to install via the Nordugrid repository: NorduGrid Repositories.

Note: Alpha and release-candidate packages are in testing repository, so please make sure it is enabled, e.g. on RHEL-based systems you can use yum --enablerepo=nordugrid-testing to enable it for one transaction or yum-config-manager --enable nordugrid-testing to enable permanently.

If you want to test ARC6 including all latest developments, set up your repository to include the nightly builds: Using ARC packages from nightly builds instructions.

Step 2. Install A-REX

ARC Resource-coupled EXecution service (A-REX) is a core component that manages authentication, authorization and job life cycle. It is enough to have A-REX installed to have a minimal computing element:

[root ~]# yum -y install nordugrid-arc-arex

Step 3. Run A-REX

To start ARC services just run:

[root ~]# arcctl service start --as-configured

You can check if A-REX is running with:

[root ~]# arcctl service list
arc-acix-index (Not installed, Disabled, Stopped)
arc-acix-scanner (Not installed, Disabled, Stopped)
arc-arex (Installed, Enabled, Running)
arc-datadelivery-service (Not installed, Disabled, Stopped)
arc-gridftpd (Not installed, Disabled, Stopped)
arc-infosys-ldap (Not installed, Disabled, Stopped)
**Note:** `arcctl` tool automates many ARC CE operations and is designed with bash-completion in mind. If you would like to use ARC in production it is advised to have completion enabled:

```
[root ~]# yum install bash-completion python-argcomplete
[root ~]# activate-global-python-argcomplete
```

### Step 4. Generate user certificate and key for testing

Grid services and users authentication heavily relies on cryptography and uses certificates/keys for each entity. ARC6 comes with Test Certificate Authority on board that can issue the test user certificates easily.

The ARC6 zero configuration implements a *default closed* approach defining the special authorization object called `authgroup`.

During the test-user certificate generation, `arcctl test-ca` will automatically add the issued certificate subject to the `testCA.allowed-subjects` file, opening the job submission possibiility to the test-user transparently. The `testCA.allowed-subjects` can be found in your `/etc/grid-security` folder.

No other subject will be able to submit to your system before you change the `authgroup` settings in `arc.conf`.

You can test submission from the host running A-REX or from any other host in the network.

#### Testing from the host running A-REX

It is technically possible to submit jobs from the `root` account, however it is advised to use a dedicated regular user. Here we assume that you use a dedicated regular user.

To generate test certificate/key and install it to standard location inside local user’s home directory run:

```
[root ~]# arcctl test-ca usercert --install-user user01
User certificate and key are installed to default /home/user01/.globus location
--for user user01.
```

#### Testing from any other host

In order to submit jobs from any other host (not the one running A-REX) you need to transfer the (test) user certificate and the CA-files to this other host.

On the A-REX host generate a user certificate/key:

```
[root ~]# arcctl test-ca usercert --export-tar
User certificate and key are exported to testcert-09160712.tar.gz.
To use it with arc* tools on the other machine, copy the tarball and run the
following commands:
tar xzf testcert-09160712.tar.gz
source arc-test-certs/setenv.sh
```

Transfer the tarball to the client host and on the client host execute the commands suggested in the `arcctl` output:

```
[user ~]$ tar xzf /tmp/testcert-09160712.tar.gz
[user ~]$ source arc-test-certs/setenv.sh
```

**Note:** The zero configured A-REX comes with EMI-ES and REST interfaces enabled. It runs on port 443, so make sure it is not firewalled to be able to be used from another client host.
Step 5. Submit job and check it is running

Install ARC client tools on the client host:

```
[root ~]# yum -y install nordugrid-arc-client
```

To submit a job, or perform any other action towards the ARC server you need a so-called *proxy-certificate* which is a Single Sign-On token for distributed grid-infrastructure. It is generated in the following way:

```
[user ~]$ arcproxy
Your identity: /DC=org/DC=nordugrid/DC=ARC/O=TestCA/CN=Test Cert 08272152
Proxy generation succeeded
Your proxy is valid until: 2018-08-28 09:54:24
```

You can start with the information query about your newly installed ARC computing element¹:

```
[user ~]$ arcinfo -c https://arc.example.org/arex
Computing service:
  Information endpoint: https://arc.example.org:443/arex
  Information endpoint: https://arc.example.org:443/arex
  Information endpoint: https://arc.example.org:443/arex
  Submission endpoint: https://arc.example.org:443/arex (status: ok, interface:
    org.ogf.glue.emies.activitycreation)
  Submission endpoint: https://arc.example.org:443/arex (status: ok, interface:
    org.ogf.bes)
  Submission endpoint: https://arc.example.org:443/arex (status: ok, interface:
    org.nordugrid.arcrest)
```

A simple job can be submitted with the *arctest* tool:

```
[user ~]$ arctest -J 2 -c https://arc.example.org/arex
Submitting test-job 2:
'( executable = "/usr/bin/env" )( stdout = "stdout" )( stderr = "stdout" )( gmlog = "gmlog" )( jobname = "arctest2" )( clientxrsl = "( executable = "/usr/bin/env" )( jobname = "arctest2" )( stdout = "stdout" )" )
Client version: nordugrid-arc-20180822231219
Test submitted with jobid: https://arc.example.org:443/arex/NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm
```

The job status can be checked with the *arcstat* tool:

```
[user ~]$ arcstat https://arc.example.org:443/arex/
--Nh1KDsmsmeEt7nPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm
Job: https://arc.example.org:443/arex/
--Nh1KDsmsmeEt7nPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm
Name: arctest2
State: Running
Status of 1 jobs was queried, 1 jobs returned information
```

To fetch the job’s stdout run *arccat* tool:

```
[user ~]$ arccat https://arc.example.org:443/arex/
--Nh1KDsmsmeEt7nPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm -f stdout
GRIDMAP=/dev/null
HOSTNAME=arc.zero
TMPDIR=/tmp
GLOBUS_LOCATION=/usr
<output omitted>
```

¹ Examples uses arc.example.org as a domain name for A-REX host
Step 6. Play more with the ARC Computing Element

As an admin you might frequently need to extract information from the logs and directories that ARC computing element uses. The brief list of the relevant paths can be obtained from:

```
[root ~]# arcctl config brief
ARC Storage Areas:
  Control directory:  
    /var/spool/arc/jobstatus
  Session directories:  
    /var/spool/arc/sessiondir
  Scratch directory on Worker Node:  
    Not configured
  Additional user-defined RTE directories:  
    Not configured
ARC Log Files:
  A-REX Service log:  
    /var/log/arc/arex.log
  A-REX Jobs log:  
    /var/log/arc/arex-jobs.log
  A-REX Helpers log:  
    /var/log/arc/job.helper.errors
  A-REX WS Interface log:  
    /var/log/arc/ws-interface.log
  Infosys Infoproviders log:  
    /var/log/arc/infoprovider.log
```

To get information and manage jobs on A-REX server, the `arcctl job` is useful. Operations include but is not limited to:

- Listing jobs:

```
[root ~]# arcctl job list
Nh1KDmsmeEt7nPSAtDmVuSEmABFKDmABFKDm2PJJKaMBFBKDbmDyQbom
zb0LDm7RfEt7nPSAtDmVuSEmABFKDmABFKDm2PJJKaMBFBKDbm1CYXm
<output omitted>
[root ~]# arcctl job list --long
Nh1KDmsmeEt7nPSAtDmVuSEmABFKDmABFKDm2PJJKaMBFBKDbmDyQbom FINISHED arctest2
   /DC=org/DC=nordugrid/DC=ARC/O=TestCA/CN=Test Cert 08272152
zb0LDm7RfEt7nPSAtDmVuSEmABFKDmABFKDm2PJJKaMBFBKDbm1CYXm FINISHED arctest2
   /DC=org/DC=nordugrid/DC=ARC/O=TestCA/CN=Test Cert 08272234
<output omitted>
```

- Job general information:

```
[root ~]# arcctl job info Nh1KDmsmeEt7nPSAtDmVuSEmABFKDmABFKDm2PJJKaMBFBKDbmDyQbom
Name          : arctest2
Owner         : /DC=org/DC=nordugrid/DC=ARC/O=TestCA/CN=Test Cert 08272152
State         : FINISHED
LRMS ID       : 16890
Modified      : 2018-08-27 22:06:34
```

- Job log:

```
[root ~]# arcctl job log Nh1KDmsmeEt7nPSAtDmVuSEmABFKDmABFKDm2PJJKaMBFBKDbmDyQbom
2018-08-27T22:03:34Z Job state change UNDEFINED -> ACCEPTED Reason: 
   (Re)Accepting new job
2018-08-27T22:03:34Z Job state change ACCEPTED -> PREPARING Reason: Starting job
   processing
2018-08-27T22:03:34Z Job state change PREPARING -> SUBMIT Reason: Pre-staging
   finished, passing job to LRMS
2018-08-27T22:03:36Z Job state change SUBMIT -> INLRMS Reason: Job is passed to
   LRMS
```

(continues on next page)
A-REX logs that mentions the job:

```
[root ~]# arctct job log NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm --
## /var/log/arc/arcex.log:
[2018-08-27 22:03:34] [Arc] [INFO] [16298/4]
-NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: State: ACCEPTED: parsing
- job description
[2018-08-27 22:03:34] [Arc] [INFO] [16298/4]
-NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: State: ACCEPTED: moving
to PREPARING
[2018-08-27 22:03:34] [Arc] [INFO] [16298/4]
-NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: State: PREPARING from
- ACCEPTED
[2018-08-27 22:03:34] [Arc] [INFO] [16298/4]
-NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: State: SUBMIT from
- PREPARING
[2018-08-27 22:03:34] [Arc] [INFO] [16298/4]
-NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: state SUBMIT: starting
-child: /usr/share/arc/submit-fork-job
[2018-08-27 22:03:36] [Arc] [INFO] [16298/4]
-NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: state SUBMIT: child
-exited with code 0
[2018-08-27 22:03:36] [Arc] [INFO] [16298/4]
-NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: State: INLRMS from SUBMIT
[2018-08-27 22:06:34] [Arc] [INFO] [16298/4]
-NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: State: FINISHED from
- INLRMS
[2018-08-27 22:06:34] [Arc] [INFO] [16298/4]
-NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm: Job finished
```

• Getting job attributes:

```
[root ~]# arctct job attr NhlKDmsmeEtnPSAtDmVmuSEmABFKDmABFKDm2PJKDmBBFKDmxDyQbm --
-jobname
arctest2
```

Now you are ready to Install production ARC6 Computing Element!

### 3.2.2 ARC Computing Element Installation and Configuration Guide

#### Prerequisites

Now you are ready to Install production ARC6 Computing Element!
Choosing the host

It is assumed that ARC CE is installed on top of an existing Linux computing cluster. Many Linux distributions are supported. ARC works well also on a complete virtual computing cluster environment in a cloud.

ARC is non-intrusive towards existing systems. We suggest to deploy ARC CE on a dedicated (virtual) machine connected to the cluster network and filesystem.

ARC software is very lightweight and does not require powerful machines to run, however if ARC CE will perform data transfers the requirements are higher. As a minimum, a production CE with 4 cores and 8GB of RAM should be capable of handling up to 10,000 concurrent jobs without problems. One CE can easily handle the load of a single cluster, however multiple CEs may be deployed in parallel for redundancy.

Plan for storage areas

Several storage areas are necessary for job submission, execution and data storing. You should mount/export following directories:

- *session directory*
- *data staging cache directory* (if planned)
- decide to what extent to use NOT cross-mounted *scratch directory* scratch directory on the worker nodes

Local resource management system (LRMS)

Install and configure your LRMS (batch system). ARC supports a variety of LRMS back-ends:

- *fork* - fork jobs on the ARC CE host node, not a cluster. Targeted for testing and development but not for real production workloads.
- *condor* - uses HTCondor-powered HTC resource
- *slurm* - for SLURM clusters
- *pbs* - any flavor of PBS batch system, including Torque and PBSPro
- *ll* - Load Leveler batch system
- *lsf* - Load Sharing Facility batch system
- *sge* - Oracle Grid Engine (formerly Sun Grid Engine)
- *boinc* - works as a gateway to BOINC volunteer computing resources

Start by checking if you are able to submit jobs to the chosen LRMS from the ARC CE host.

You may consider setting up dedicated queues to use with ARC CE (e.g. per-VO queues).

Please also NOTICE that in some cases (*depending on LRMS*) you need to share the batch system log directories with ARC CE.

Configure OS accounts

Plan for local account(s) (or account pools) that will be used to execute jobs on the worker nodes.

These accounts should be also available on the ARC CE node.

Please note that ARC services are ran as root on the ARC CE node and switch to an appropriate local account when processing job data staging and job execution. This process is called *mapping*. 
Installation

This section assumes you have already enabled the NorduGrid repositories for your package utility (yum/dnf/apt).

Install ARC CE core packages from repositories:

```
[root ~]# yum -y install nordugrid-arc-arex
or
[root ~]# apt-get install nordugrid-arc-arex
```

Any extra packages will be installed based on the ARC configuration file with ARC Control Tool as described below. Full list of packages to install manually (especially additional plugins) can be found here.

Grid security heavily relies on PKI and all actions requires certificates/keys for ARC CE as a service and users:

- for testing purposes, a dummy Test-CA and host certificates signed by the Test-CA are generated during A-REX installation.
- for production use please obtain a certificate signed by one of the IGTF accredited CAs and remove Test-CA files with arcctl test-ca cleanup.

ARC CE needs IGTF CA certificates deployed to authenticate users and other services, such as storage elements. To deploy IGTF CA certificates to ARC CE host, run¹:

```
[root ~]# arcctl deploy igtf-ca classic
```

Configuration

Configuration of ARC CE can be done by means of modifying the pre-shipped ‘zero configuration’ available at /etc/arc.conf.

The purpose of this ‘zero configuration’ is to offer a minimalistic working computing element out-of-the box right after package installation with zero additional configuration needed.

For production deployment you will need to customize the configuration in accordance to your actual setup and operations mode.

**Note:** ARC services must be restarted when changes have been made to arc.conf.

The ultimate information about available configuration options can be found in the ARC Configuration Reference Document which is also available locally as /usr/share/doc/nordugrid-arc-*/*/arc.conf.

The most common configuration steps are explained below.

Configure authorization and mapping rules

Authorization rules define who can access the computing element (execute jobs, query info, etc). Mapping rules define which grid-users are mapped to which system accounts.

Both authorization and mapping rules in ARC6 rely on the concept of authgroups. Each authgroup represents a set of users, whose identities are matched to configured rules.

Once defined, authgroups can be applied to filter access to the CE per interface ([arex/ws/jobs], [gridftpd/jobs]) or per-queue.

The allowaccess and/or denyaccess options in the corresponding block define which authgroups are allowed to access the interface or submit to the queue.

¹ Use --installrepo argument to enable repositories with IGTF CA certificates if ARC is not installed from the NorduGrid repos.
The [mapping] block used to configure the rules that defines how the particular authgroup members are mapped to OS accounts.

In the shipped zero configuration the [authgroup: zero] is defined and applied to A-REX WS interface, the effect of which is to deny any access unless user is listed in the testCA.allowed-subjects file. The mapping is configured with map_to_user rule that assign the same nobody account to everyone in zero authgroup.

The typical configuration looks like this:

```
[authgroup: atlas]
voms = atlas * * *

[mapping]
map_to_pool = atlas /etc/grid-security/pool/atlas

[gridftp/jobs]
allowaccess = atlas

[queue: qatlas]
allowaccess = atlas
```

Please read the Authorization and Mapping rules in ARC6 document to get familiar with all aspects of this important configuration step.

Provide LRMS-specific information

One more critical configuration step is to supply ARC CE with relevant information regarding your LRMS specifics.

Specify your LRMS type

In the arc.conf there is a dedicated [lrms] block that defines the type of your LRMS, as well as several options related to the LRMS behaviour. For example, to instruct ARC to use SLURM, use the following configuration:

```
[lrms]
lrms = slurm
slurm_use_sacct = yes
```

Specify queues

In addition to specifying LRMS itself, it is necessary to list all the queues that will be exposed via the ARC CE, by using [queue: name] blocks.

```
[queue: atlas]
comment = Queue for ATLAS jobs
```

More information about configuring particular LRMS to work with ARC can be found in Batch systems support document.

Configure A-REX Subsystems

The ARC Resource-coupled EXecution service (A-REX) is a core service handling execution and entire life cycle of compute jobs.
Enable job management interfaces

A-REX has several job management interfaces available. One can control which of them are enabled and exposed by configuring the corresponding blocks.

**WS Interfaces (EMI-ES and ARC REST)**  
`[arex/ws/jobs]`

**Gridftp**  
`[gridftp/jobs]`

**Internal**  
Install `nordugrid-arc-plugins-internal` package to use this interface.

Enable data services

ARC has a built-in data transfer framework called DTR. It was designed to be used in environments in which data transfer was not possible or not desirable on the worker nodes such as HPC centres or sites without local storage.

DTR relies on users submitting jobs with pre-defined input and output files. When A-REX receives a job, it takes care of downloading the specified input files to the job’s session directory, then submits the job to the batch system. After the batch job finishes, A-REX takes care of uploading any output files to grid storage.

Define the `[arex/data-staging]` block to enable data-staging capabilities. Data transfers can be scaled out using `multi-host data-staging`.

DTR also includes a caching capability. If caching is enabled then A-REX will download all input files to the cache, and create symlinks from the session directory for each file. If a job requests a file that is already cached, A-REX will not download it again, but simply link from the existing cache file. Define the `[arex/cache]` block to enable caching.

More detailed technical documentation on ARC data features and advanced features such as ACIX and Candy-Poind can be found in the data overview pages.

RunTime Environments

RunTime Environments can modify the job execution cycle and are used for advertising software or features offered by the computing facility.

ARC ships several RTEs that are ready to be used and classified as system-defined.

One can add ones own directories with so-called user-defined RTEs using the `runtimedir` configuration option in the `[arex]` block.

In ARC6, both system- and user-defined directories are local to the ARC CE node and SHOULD NOT be shared to worker nodes (unlike in ARC 5).

To use an installed RTE, one should additionally **enable** this RTE with `ARC Control Tool`. For example, to enable the system-defined `ENV/PROXY` RTE, run:

```
[root ~]# arcctl rte enable ENV/PROXY
```

More details on operating RunTime Environments can be found in *RunTime Environments in ARC6*.

Information system

ARC CE information system aims to collect and publish information to be used by special clients for matchmaking and/or monitoring the state and stats of the resource.

It is mandatory to configure the information system for production cases, like those of the WLCG computing infrastructure.
Defining general information

There are many information schemas and renderings of data available to comply to existing standards. There are several blocks that are used to define published information depending on schemas:

[infosys] The most common block that enables internal information collection from ARC CE host and LRMS.

[infosys/cluster] The common information about the whole cluster, including e.g. its capacity.

[queue: name] For heterogeneous clusters, most of the information in the [infosys/cluster] block can be re-defined on per-queue basis.

[infosys/glue2] Configures the GLUE2-specific values and enables internal glue2 rendering.

[infosys/ldap] Enables LDAP/BDII dedicated services to publish information via LDAP protocol.

[infosys/glue2/ldap] Enables GLUE2-schema LDAP rendering of the information.

[infosys/nordugrid] Enables LDAP rendering of the information according to the NorduGrid schema.

[infosys/glue1] Configures the GLUE1.x-schema specific values and enables LDAP rendering of GLUE1.x.

[infosys/glue1/site-bdii] Enables and configures GLUE1.x site-bdii functionality.

Accounting

ARC CE has built-in functionality to publish usage statictics to the SGAS and APEL centralized accounting services using the Job Usage Reporter of ARC (JURA) component.

JURA creates standard-compliant usage records from job usage information provided by the A-REX Job Log files, send the records to remote accounting services and optionally archive the records for future analyses and republishing.

If you need to configure accounting follow the Accounting with JURA guide.

Configure Firewall

Different ARC CE services open a set of ports that should be allowed in the firewall configuration.

To generate iptables configuration based on arc.conf, run:

```
[root ~]# arcctl deploy iptables-config
```

Enable and Run Services

To enable and run all services as configured in arc.conf, run:

```
[root ~]# arcctl service enable --as-configured --now
```

Instead of using ARC Control Tool to manage ARC services, you can always use your OS native tools.

Test Basic Functionality

To test some basic job submission to the configured ARC CE, follow the instructions provided in the Try ARC6: towards distributed computing in a few minutes.
3.2.3 ARC5 to ARC6 Migration Guide

ARC 6 comes with major re-engineering in the code base leading to improvements in the deployment, configuration, maintenance and scalability of ARC CE. It introduces new features while some legacy interfaces, components and features have been obsoleted and removed.

Despite all the code changes the supported ARC 6 CE interfaces are unchanged as well as the structure of the control directory, session directory, accounting logs, etc. This makes it possible to upgrade ARC5 to ARC6 without a complete re-installation “from scratch”.

Note: Despite the fact that we haven’t observed any issues during normal package upgrades it is recommended to perform a clean install “from scratch” wherever possible in order to eliminate hidden bugs and ARC5 leftovers.

Following sections provides you with the necessary info about the major steps of an ARC5 to ARC6 migration.

Quick reference

Here is a quick 6 step checklist for what to do:

1. Rewrite your arc.conf: you can send your configuration to us and we will help to rework it for ARC6.
2. Learn the ARC6 packages and services names.
3. Get familiar with ARC Control Tool that will simplify your life as an ARC6 administrator.
4. Don’t forget to enable RunTime Environments (RTE) with arcctl rte enable.
5. Be aware of accounting subsystem changes.
6. Say “Good Bye!” to EGIIS, the ARC community is migrating over to a DNS-based ARCHERY system.

The details:

Configuration file

Because of the complete overhaul of the server-side ARC CE configuration, an existing ARC5 configuration can not be used with an ARC6 installation, it will immediately fail during the configuration validation process of the ARC service startup. Therefore you must create a new ARC6 configuration from scratch.

Note: There is no automatic converter from an old ARC5 configuration file to the new ARC6 syntax. The main reason for this is that there are many conceptual changes rather than simple option and block renaming changes.

We offer support in configuration rework via nordugrid-discuss mailing list or ARC6 Support Skype Channel. Just send your configuration to us!

When reworking the configuration take notice of these syntax changes:

• no more quotes in option values
• blocks are now not only grouping options, but have a enable/disable functionality
• in block names keywords are separated by / to represent subblocks (if such exist), while block identifiers are separated by : from keywords

Note: It is advised to read the General configuration structure section of ARC configuration reference document before starting any rewrite.

There are also changes in the configuration approach in several areas. The major highlights are:
• The way authorization and user mapping is handled has been greatly reworked and improved. In particular mapping via gridmap files is NO longer required nor recommended. If you still have gridmap files in your configuration consider whether you really want to keep this approach in ARC6.

• Jura accounting configuration is now configured within dedicated blocks.

• Infosys schemas are now also enabled/disabled with blocks. In particular the default ARC5 NorduGrid LDAP schema should be explicitly enabled with [infosys/nordugrid] block.

Warning: ARC6 configuration also comes with consistent defaults for everything. If a block is enabled all configuration options in this block that has no specific value in arc.conf get their default values.

Configuration with all defaults included can be printed with arcctl config dump.

Notice that any changes in configuration requires restart off all affected ARC services.

Note: Config validation is enforced during A-REX start or can be run manually with arcctl config validate. So don’t worry if you had misspelled something, the validator will catch it!

Operating ARC services

ARC6 also comes with the a rewised consistent packages and services naming scheme. We made the service and package names more clear and consistent with configuration block names in arc.conf.

Log file locations and internal format have been reviewed to provide consistency across the ARC subsystems. Log file locations can be printed with arcctl config brief.

ARC Control Tool

To simplify common ARC CE operations, ARC6 introduces ARC Control Tool - the single entry point to administrate and work with various ARC and third-party components. The arcctl tool helps ARC site admins by wrapping typical complex operations and unifying ARC CE administration under a single utility.

The arcctl tool was designed with BASH-completion in mind. It completes subsystem names, arguments, jobIDs, certificate DNs, RTE names, etc.

Note: To get familiar with some of the arcctl functionality it is advised to follow Try ARC6: towards distributed computing in a few minutes tutorial. Using the arcctl tool to administrate an ARC6 site is recommended even for the most experienced ARC5 admins.

RunTime Environments

The new framework for RunTime Environments (RTE) has gotten a redesigned architecture and a largely extended functionality (default, installed, enabled RTEs, introduction of RTE parameters).

The major conceptual and behaviour differences are:

• in ARC6 RTE directories are local to the ARC CE head node and SHOULD NOT be shared to worker nodes

• in ARC6 the RTE concept got extended by the introduction of RTE types such as default, installed, enabled, dummy, etc. An ordinary ARC5 RTE would correspond to an installed user-defined ARC6 RTE. Please read the documentation.

• All installed ARC5 RTEs (available in a directory) should be enabled explicitly (e.g. arcctl rte enable ENV/PROXY) to be used on an ARC6 CE.
Note: You can use wildcards to enable all RTEs you have in one shot: `arcctl rte enable '*'`

**Accounting**

The ARC6 *Jura accounting* has a redesigned archive structure that introduces a database for local stats and comes with a redesigned record republishing to accounting services, in addition to the already mentioned configuration changes.

To make ARC6 accounting compatible with ARC5 *Jura* and archive structure, an automatic conversion is done by an additional *jura-archive-manager* process run by AREX.

**Warning:** The JURA archive conversion consumes CPU and I/O resources of your ARC CE. If you are upgrading and have a lot of records in archive already expect an additional load during conversion.

**Retiring the EGIIS**

All configuration blocks and services in ARC5 that correspond to registration to EGIIS are now gone in ARC6. *ARCHERY*, the new DNS-based service endpoint catalogue for ARC is replacing EGIIS and it works *top-to-bottom* without explicit registration, i.e. nothing registration related needs to be configured in the arc.conf of an ARC6 CE.

If your CE had been previously registered in the country-level nordugrid EGIIS your ARC CE registration is already migrated to the nordugrid.org ARCHERY instance.

If you find your CE missing (e.g. in the nordugrid top-level monitor) then contact your country representatives or write to us directly via mailing list or Skype channel mentioned above.

**3.2.4 ARC CE Deployment Scenarios**

To be written deployment guides:

- WLCG Deployment with Data Capabilities
- WLCG Pilot-mode Deployment
- Using ARC beyond the WLCG Scope

**3.2.5 Operating ARC CE Subsystems**
## ARC6 Packages

Table 3.1: List of ARC 6 binary packages

<table>
<thead>
<tr>
<th>Package info</th>
<th>ARC 6 package name</th>
<th>Block name</th>
<th>ARC 5 package name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base package holding common files</td>
<td>nordugrid-arc</td>
<td>N/A</td>
<td>nordugrid-arc</td>
</tr>
<tr>
<td>ARC Hosting Environment Daemon</td>
<td>nordugrid-arc-hed</td>
<td>N/A</td>
<td>nordugrid-arc-hed</td>
</tr>
<tr>
<td>ARC Resource-coupled EXecution service (A-REX)</td>
<td>nordugrid-arc-arex</td>
<td>[arex]</td>
<td>nordugrid-arc-arex</td>
</tr>
<tr>
<td>ARC Candypond Service</td>
<td>part of</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARC GridFTP Server</td>
<td>nordugrid-arc-gridftpd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARC LDAP-based Information Services</td>
<td>nordugrid-arc-infosys-ldap</td>
<td>[ldap]</td>
<td>nordugrid-arc-aris,nordugrid-arc-ldap-infosys</td>
</tr>
<tr>
<td>ARC Data Delivery Service (DDS)</td>
<td>nordugrid-arc-datadelivery-service</td>
<td>[datadelivery-service]</td>
<td></td>
</tr>
<tr>
<td>ARC Cache Index (ACIX) - Core</td>
<td>nordugrid-arc-acix-core</td>
<td>N/A</td>
<td>nordugrid-arc-acix-core</td>
</tr>
<tr>
<td>ARC Cache Index (ACIX) - Scanner</td>
<td>nordugrid-arc-acix-cache</td>
<td>[acix-cache]</td>
<td>nordugrid-arc-acix-cache</td>
</tr>
<tr>
<td>ARC Cache Index (ACIX) - Index</td>
<td>nordugrid-arc-acix-index</td>
<td>[acix-index]</td>
<td>nordugrid-arc-acix-index</td>
</tr>
<tr>
<td>The nordugridmap tool</td>
<td>nordugrid-arc-nordugridmap</td>
<td>[nordugridmap]</td>
<td>nordugrid-arc-nordugridmap</td>
</tr>
<tr>
<td>ARC development files</td>
<td>nordugrid-arc-devel</td>
<td>N/A</td>
<td>nordugrid-arc-devel</td>
</tr>
<tr>
<td>Python 2 bindings for ARC</td>
<td>python2-nordugrid-arc</td>
<td>N/A</td>
<td>python2-nordugrid-arc</td>
</tr>
<tr>
<td>Python 3 bindings for ARC</td>
<td>python3-nordugrid-arc</td>
<td>N/A</td>
<td>python3-nordugrid-arc</td>
</tr>
<tr>
<td>ARC command line clients</td>
<td>nordugrid-arc-client</td>
<td>own config</td>
<td>nordugrid-arc-client</td>
</tr>
<tr>
<td>ARC test tools</td>
<td>nordugrid-arc-test</td>
<td>N/A</td>
<td>nordugrid-arc-misc-utils</td>
</tr>
<tr>
<td>ARC LDAP monitor web application</td>
<td>nordugrid-arc-monitor</td>
<td>N/A</td>
<td>nordugrid-arc-monitor</td>
</tr>
<tr>
<td>ARC base plugins (MCCs and DMCs)</td>
<td>nordugrid-arc-plugins-needed</td>
<td>N/A</td>
<td>nordugrid-arc-plugins-needed</td>
</tr>
<tr>
<td>ARC Globus plugins</td>
<td>nordugrid-arc-plugins-globus</td>
<td>N/A</td>
<td>nordugrid-arc-plugins-globus</td>
</tr>
<tr>
<td>ARC xrootd plugins</td>
<td>nordugrid-arc-plugins-xrootd</td>
<td>N/A</td>
<td>nordugrid-arc-plugins-xrootd</td>
</tr>
<tr>
<td>ARC GFAL2 plugins</td>
<td>nordugrid-arc-plugins-gfal</td>
<td>N/A</td>
<td>nordugrid-arc-plugins-gfal</td>
</tr>
<tr>
<td>ARC S3 plugins</td>
<td>nordugrid-arc-plugins-s3</td>
<td>N/A</td>
<td>nordugrid-arc-plugins-s3</td>
</tr>
<tr>
<td>ARC Internal plugin</td>
<td>nordugrid-arc-plugin-inter</td>
<td>N/A</td>
<td>nordugrid-arc-plugin-inter</td>
</tr>
<tr>
<td>ARCHERY administration tool</td>
<td>nordugrid-arc-archery</td>
<td>N/A</td>
<td>nordugrid-arc-archery</td>
</tr>
<tr>
<td>A-REX Python LRMS backends</td>
<td>nordugrid-arc-python-lrms</td>
<td>N/A</td>
<td>nordugrid-arc-python-lrms</td>
</tr>
<tr>
<td>ARC optional worker nodes components [6,2]</td>
<td>nordugrid-arc-wn</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

List of packages deprecated in ARC6:

- `nordugrid-arc-arc-mon

- `nordugrid-arc-ws-monitor

- `nordugrid-arc-arcproxyalt

- `nordugrid-arc-ca-utils

- `nordugrid-arc-eglis

- `nordugrid-arc-java

---

1 Block names are used by `arcctl service enable --as-configured` command to fetch necessary packages automatically based on the blocks configured in `arc.conf`

2 Package define LDAP/BDII/Glue-Schema dependencies and contains wrappers to start all this LDAP world. Infoproviders are in the A-REX package.

3 No longer relevant `saml_assertion_init` tool had been removed.

4 Support for specific protocols is provided by separate 3rd-party GFAL2 plugin packages.
ARC6 Services

Table 3.2: List of ARC 6 services

<table>
<thead>
<tr>
<th>Block name</th>
<th>ARC 6 service name</th>
<th>Main process</th>
<th>ARC 5 service name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[arex]</td>
<td>arc-arex</td>
<td>arched</td>
<td>a-rex</td>
</tr>
<tr>
<td>[arex/ws/</td>
<td>started by arc-arex</td>
<td>arched</td>
<td>arc-candypond</td>
</tr>
<tr>
<td>candypond]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[gridftpd]</td>
<td>arc-gridftpd</td>
<td>gridftpd</td>
<td>gridftpd</td>
</tr>
<tr>
<td>[infosys/</td>
<td>arc-infosys-ldap</td>
<td>slapd,</td>
<td>nordugrid-arc-aris</td>
</tr>
<tr>
<td>ldap]</td>
<td></td>
<td>bdii-update</td>
<td></td>
</tr>
<tr>
<td>[datadelivery</td>
<td>arc-datadelivery-service</td>
<td>arched</td>
<td></td>
</tr>
<tr>
<td>[acix-scanner]</td>
<td>arc-acix-scanner</td>
<td>twistd</td>
<td>acix-cache</td>
</tr>
<tr>
<td>[acix-index]</td>
<td>arc-acix-index</td>
<td>twistd</td>
<td>acix-index</td>
</tr>
</tbody>
</table>

Authorization and Mapping rules in ARC6

Overview

ARC CE authorization and mapping rules rely on the concept of authgroups (configured by [authgroup] blocks).

During the connection establishment and authentication process on any of the ARC CE interfaces (e.g. job submission on the figure), the Authgroup matching happened. Each authgroup represents a set of users, whose identities are matched to configured rules. The same user can match (belongs to) several authgroups.

If configured by allowaccess and denyaccess options, per-interface authorization rules will be enforced. This rules defines the list of authgroups that granted or denied access to the particular ARC CE interface. Without per-interface authorization rules, configuration access to the interface is granted to anyone who passed authentication process.

1 Block names are used by arctl service start --as-configured command to start/stop necessary services automatically based on the blocks configured in arc.conf
Warning: User will be successfully authenticated only in case of certificate and proxy-certificate has been passed validation.

In case of time syncronization problems, missing or invalid CA certificates and CRLs, missing VOMS LSC files, etc - the connection WILL NOT be established.

Mapping rules in ARC6 also relies on authgroup membership. Configured mapping rules are processed sequentially and define the OS account ID that will be assigned to authgroup members. Several mapping methods are available, depending on requirements.

Notice that in ARC6 authorization rules can also be enforced per-queue. It uses the same allowaccess and denyaccess options syntax as for per-interface configuration but can additionally restrict access to the particular queue. By default access to the queue granted to all users authorized on interface level.

Defining authgroups

Authgroups should be defined on the top of arc.conf using the [authgroup:groupname] block before it will be referenced in the other parts of configuration.

Each authgroup is a named object, that will be referenced by its name during authorization and mapping rules configuration. There are no special restrictions to the authgroup names except the absence of spaces, so you can even define * authgroup to blow the mind of other arc.conf readers.

Each config line in the [authgroup:groupname] block represent a matching rule that are processed sequentially.

When the matchig criteria of the rule has been satisfied by user identity - the processing stops within this authgroup.

Whether user belogns to this authgroup or not is defined by the type of rule that was matched: there are positively and negatively matching rules. By default all rules are positive (user IS a member of authgroup when matched) unless prefixed with – sign.

Matching one of the authgroups does not mean that the same user is not processed for the next authgroup. All authgroups are evaluated, even if a user already has a match with one of the earlier processed authgroups.

Note: Notice that:

- authgroup blocks should be defined before referencing!
- authgroup rules within blocks are order-dependent!
- all authgroup blocks are evaluated!

Complete list of authgroup rules can be found in the ARC Configuration Reference document. Some examples are:

```
[authgroup: norduguest]
-file = /etc/grid-security/banned.dns
voms = nordugrid Guests * *

[authgroup: admin]
subject = /O=Grid/O=Big VO/CN=Main Admin

[authgroup: lcas]
plugin = 10 /usr/libexec/arc/arc-lcas %D %P liblcas.so /usr/lib64 /etc/lcas/lcas.db

[authgroup: any]
authgroup = norduguest
authgroup = admin
authgroup = lcas
```

(continues on next page)
Applying authorization rules

You can enforce authentication restrictions on every interface. This includes:

- EMI-ES and REST interfaces configured with `[arex/ws/jobs]` block
- GridFTP job submission interface configured with `[gridftpd/jobs]` block
- GridFTP trivial storage interface configured with `[gridftpd/filedir]` block

In addition to interface level authorization, queue-level authorization can be configured using the same configuration approach.

The `allowaccess` configuration option defines that the specified `authgroup` members are authorized to access the ARC-CE via this interface or access particular queue. A related config option `denyaccess` can in turns be used to reject access.

Multiple `allowaccess` and `denyaccess` authorization statements are allowed within a configuration block. These statements are processed sequentially, in the order they are specified in the config block.

The processing stops on first `allowaccess` or `denyaccess` statement matching the authgroup membership. If there are no authorization statements specified, then no additional restrictions are applied for authorizing user access and the interface or queue is open to everybody authenticated.

**Note:** Default deny authorization approach can be configured using the empty authgroup.

Example: ARC6 zero configuration

In the shipped zero configuration the `[authgroup: zero]` is defined and applied to A-REX WS interface. The effect of this configuration is to allow access to CE only to the subjects stored in the `testCA.allowed-subjects` file. This file is empty by default and close down CE access until subjects are added by `arcct1 test-ca usrcert`.

```plaintext
[authgroup: zero]
file = /etc/grid-security/testCA.allowed-subjects

[arex/ws/jobs]
allowaccess = zero
```

Example: subject-based authorization

To authorize users based on certificate subject the `subject` or `file` rules can be used.

The `file` option support both:

- plain list of subjects (each line contains only a subject name),
- grid-mapfile format, when subject name followed by mapped account ID.

In both cases subject name should be enquoted if it contains spaces.
Example: VOMS-based authorization

To filter access based on VOMS certificate attributes, define one or more [authgroup] blocks using the voms keyword.

To verify VO membership signatures, ARC CE needs the so-called list of certificates (LSC) files that can be installed by arcctl.

Example configuration for atlas and alice VO:

1. Deploy LSC files:

```bash
[root ~]# arcctl deploy voms-lsc atlas --egi-vo
[root ~]# arcctl deploy voms-lsc alice --egi-vo
```

2. Create authorization group and apply access restriction to interface and/or queue in arc.conf:

```plaintext
[authgroup: atlas]
voms = atlas * * *

[authgroup: alice]
voms = atlas * * *

[authgroup: all]
authgroup = atlas
authgroup = alice

[gridftpd/jobs]
allowaccess = all

[arex/ws/jobs]
allowaccess = all

[queue: qalice]
allowaccess = alice

[queue: qatlas]
allowaccess = atlas
```

Configure mapping

Any grid user should be mapped to a local account to start processes and access files.

---

In this example and in what follows, a simplified configuration is shown. An actual configuration will in most cases include different authgroups for different VO groups and roles.
Mapping rules configured in [mapping] block define which grid-users (specified by authgroup) are mapped to which system accounts (several mapping methods available).

Rules in the [mapping] block are processed in a sequence in line order of the configuration file (from top to bottom).

There are two kind of rules available:

- mapping rules (started with map_) that defines how the particular authgroup members are mapped,
- policy rules (started with policy_) that modifies the mapping rules sequence processing.

Default policy for mapping rules processing is:

- processing CONTINUES to the next rule if identity of user DO NOT match authgroup specified in the rule (can be redefined with policy_on_nogroup option)
- processing STOPS if identity of user matched the authgroup specified in the mapping rule. Depend on whether this mapping rule returns valid UNIX identity the processing can be redefined with policy_on_map and policy_on_nomap options.

Policy can be redefined at the any point of configuration sequence and affects all mapping rules defined after the policy rule.

**Warning:** If mapping process STOPS and there is still no local UNIX identity identified, the user running A-REX will be used (typically root unless redefined by user option for specific deployment case).

When grid-identity is mapped to root account - request processing fails implicitly!

### Example: mapping to the same account

The map_to_user option allows to map all authgroup members to the same account specified as an argument.

For example in shipped zero configuration all users that are matched to authgroup zero are mapped to the same nobody account (and nobody group) that will work with local job forking:

```plaintext
[mapping]
map_to_user = zero nobody:nobody
```

### Example: mapping to the accounts pool

The most secure and flexible way is to map authgroup members to account pools (so-called map_to_pool method). It is recommended to use pools mapping when the resource is under the use of different communities.

In this approach, every member of specified authgroup will be dynamically mapped to one of the available accounts in the configured pool.

Available pool account names are stored one per line in the pool file inside the dedicated directory. Accounts from pool are assigned by means of leasing approach. All leased accounts are stored in the other files placed in the same directory. They can be reassigned to other users after 10 days of inactivity.

**Example configuration for atlas:**

1. Create necessary number of accounts to be used on ARC CE and Worked Nodes of the cluster.
2. Define ARC accounts pool:

```plaintext
[root ~]# mkdir -p /etc/grid-security/pool/atlas
[root ~]# for u in atlas{001..100}; do echo $u >> /etc/grid-security/pool/atlas/pool; done
```
2. Configure mapping in `arc.conf`:

```plaintext
[mapping]
map_to_pool = atlas /etc/grid-security/pool/atlas
```

Example: Legacy grid-mapfile based mapping

**Warning:** Legacy grid-mapfile based mapping is **NOT recommended** for the typical production loads.

In the grid-mapfile approach users are mapped to local accounts based on certificate DNs only. Mapping rules are stored line-by-line in the so-called grid-mapfile that describes which user is mapped to which account, for example:

```
"/O=Grid/O=NorduGrid/OU=uio.no/CN=Aleksandr Konstantinov" user1
"/O=Grid/O=NorduGrid/OU=hep.lu.se/CN=Oxana Smirnova" user2
```

In the simplest legacy case ARC can use the grid-mapfile for both authorization and mapping decisions.

**Example configuration for legacy grid-mapfile case:**

```plaintext
[authgroup: legacy]
file = /etc/grid-security/grid-mapfile

[mapping]
map_with_file = legacy /etc/grid-security/grid-mapfile
```

Grid-mapfiles in `arc.conf` can be also referred as a `userlist` objects and be generated regularly, keeping them up-to-date (from e.g. VOMS database) with `nordugridmap` utility that can be used and configured with the `nordugridmap` block.

**Note:** You can find more information about moving from grid-mapfiles in the *The life without gridmapfiles* presentation.

**Example: mapping with external LCMAPS rules**

ARC can run an external plugin to map users that can be configured with the `map_with_plugin` option.

To support several production loads, ARC ships with the built-in LCMAPS plugin included in A-REX package:

```plaintext
[authgroup: all]
all = yes

[mapping]
map_with_plugin = all 30 /usr/libexec/arc/arc-lcmaps %D %P liblcmaps.so /usr/lib64 \-
/etc/lcmaps/lcmaps-arc-argus.db arc
```

LCMAPS itself is a third-party tool that should be installed and configured separately, which is beyound the scope of this guide.

**Batch systems support**

`atlas` is the name used in `[authgroup: atlas]`
Overview

The A-REX has to interface to the LRMS in order to be able to submit jobs and query their information. The A-REX supports several Local Resource Management Systems (LRMS), with which it interacts by several backend scripts.

The A-REX assumes that the LRMS has one or more *queues*, which is a couple of (usually homogeneous) worker nodes grouped together. The different LRMSes have different concepts of queues (or have no queues at all).

Nevertheless, in the A-REX configuration, the machines of the LRMS should be mapped to A-REX queues. The client side job submission tools query the information system for possible places to submit the jobs, where each queue on a CE is represented as an *execution target*, and treated separately.

Configuring A-REX to use one of these LRMS backends typically involves the following steps:

1. Sharing directories between A-REX, the LRMS frontend and its working nodes. It might involve setup of shared filesystems such as NFS or similar.
3. Configuring the A-REX in respect to the *shared scratch* directories configuration.

![LRMS frontend and the nodes sharing the session directory and the local users](image)

**Fig. 3.2:** The LRMS frontend and the nodes sharing the session directory and the local users

General LRMS configuration

In the `[lrms]` block he name of the LRMS has to be specified with the `lrms` option.

The supported LRMS are:

- fork - fork jobs on the ARC CE host node, not a cluster. Targeted for testing and development but not for real production workloads.
- condor - uses HTCondor-powered HTC resource
• slurm - for SLURM clusters
• pbs - any flavor of PBS batch system, including Torque and PBSPro
• ll - Load Leveler batch system
• lsf - Load Sharing Facility batch system
• sge - Oracle Grid Engine (formerly Sun Grid Engine)
• boinc - works as a gateway to BOINC volunteer computing resources
• slurmpy - new experimental SLURM backend written in Python (requires `nordugrid-arc-python-lrms` package installed).

Each LRMS has its own specific configuration options that are prefixed with LRMS name in `[lrms]` block.

Beside this specific options, the behaviour of LRMS backend is affected by storage areas and limits setup, in particular:

• `tmpdir` - defines the path to directory for temporary files on the worker nodes
• `shared_filesystem`, `scratchdir` and `shared_scratch` - changes the way how A-REX will store the jobs’ data during the processing. More details can be found in *Job scratch area* document.
• `defaultmemory` and `req-queue defaultmemory` - set the memory limit values for jobs that has no explicit requirements in the job description.

**Accounting considerations**

A-REX have two simultaneously running approaches to collect accounting information:

• using data provided by LRMS
• using measurement from the GNU Time utility that wraps the job executable invocation inside the job script

Depending on LRMS type in use there are different kind of information available and/or missing in the LRMS accounting subsystem. It is recommended to have the GNU Time utility installed on the worker nodes to have resources measurements in all cases.

To enable GNU time usage you should define `gnu_time` option in the `[arex]` block pointing to the utility path, e.g. `/usr/bin/time`.

**Fork Backend**

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.

For is the default backend used in ARC pre-shipped zero configuration.

**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.

It is still requires queue definition and the queue should be named `fork`.

Example:

```
[lrms]
lrms = fork

[queue:fork]
```
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.

Portable Batch System (PBS)

The Portable Batch System (PBS) is one of the most popular batch systems for small clusters. 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, one of the max_user_run or max_running attributes MUST be set and its value SHOULD BE IN AGREEMENT with the number of available resources (i.e. don’t set the max_running = 10 if there are only six (virtual) processors in the system). If both max_running and max_user_run are set 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 a 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 correctly.

6. PBS server logs SHOULD BE shared with ARC CE to allow backend scripts to check the job status and collect information needed for accounting. The path to logs on the ARC CE is defined with pbs_log_path option.

Additional useful configuration hints:

• If possible, please use queue-based attributes instead of server level ones.

• The acl_user_enable = True attribute may be used with the 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.

• Node properties from the <PBS home on the server>/server_priv/nodes file together with the resources_default.neednodes can be used to assign a queue to a certain type of node.

Checking the PBS configuration:
• The node definition can be checked by pbsnodes -a. All the nodes MUST have ntype=cluster.

• The required queue attributes can be checked as qstat -f -Q queuename. There MUST be a max_user_run or a max_running attribute listed with a REASONABLE value.

Example:

```
[lrms]
lrms = pbs
defaultmemory = 512
pbs_log_path = /net/bs/var/log/torque/server_logs

[queue:grid_rt]
comment = Realtime queue for infrastructure testing
allowaccess = ops
advertisedvo = ops

[queue:alien]
comment = Dedicated queue for ALICE
allowaccess = alice
advertisedvo = alice
defaultmemory = 3500
```

**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.

**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.

For production use-cases it is recommended to enable slurm_use_sacct option.

Example:

```
[lrms]
lrms=slurm
slurm_use_sacct=yes
defaultmemory=4096

[queue:normal]
comment=Queue for grid jobs
architecture=x86_64
totalcpus=1500
```

**Using Python LRMS backend implementation**

Experimental python LRMS backend can be used for SLURM after nordugrid-arc-python-lrms package installation. Python backed is distinguished by slurmpy name that should be specified in lrms option.
This backend respects the same options set, as a classical SLURM backend script, but additionally allows the connection over SSH when [lrms/ssh] block is enabled and configured.

**Known limitations**

If you have nodes with different amount of memory in the same queue, this will lead to miscalculations. If SLURM is stopped, jobs on the 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.

**HTCondor**

The HTCondor 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 HTCondor on the A-REX node and configure it as a submit machine. Next, add the following to the node’s Condor configuration (or define CONDOR_IDS as an environment variable):

```
CONDOR_IDS = 0.0
```

`CONDOR_IDS` has to be 0.0, so that Condor will be run as root and can then access the Grid job’s session directories (needed to extract various information from the job log).

Make sure that no normal users are allowed to submit Condor jobs from this node. If normal user logins are not allowed on the A-REX machine, then nothing needs to be done. If for some reason users are allowed 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:

```
[root ~]# chgrp griduser $condor_bin_path/condor_submit
[root ~]# chmod 750 $condor_bin_path/condor_submit
```

Example:

```
[lrms]
lrms = condor
defaultmemory = 2000

[queue:EL7]
comment = EL7 queue
defaultmemory = 3000
nodememory = 16384
condor_requirements = (Opsys == "linux") && (OpSysMajorVer == 66)
```

**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.
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.

Known limitations

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

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 a convenient alternative.

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

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RES_REQ = type==X86_64</code></td>
</tr>
<tr>
<td><code>RES_REQ = type==any</code></td>
</tr>
</tbody>
</table>

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

The `lsf_profile_path` option 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.
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 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 it is desired 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. You must also make sure that the ARC CE can run qacct, as this is used to supply accounting information.

Example:

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

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

**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-maxqueueable 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.

BOINC

BOINC is an open-source software platform for computing using volunteered resources. Support for BOINC in ARC is currently at the development level and to use it may require editing of the source code files to fit with each specific project.

**Recommended batch system configuration**

The BOINC database can be local to the ARC CE or remote. Read-access is required from the ARC CE to check for finished jobs and gather information on available resources. The ARC CE must be able to run commands in the project’s bin/ directory.

Project-specific variables can be set up in an `RTE` which must be used for each job. The following example shows the variables which must be defined to allow job submission to BOINC for the project “example” to work:

```
export PROJECT_ROOT="/home/boinc/project/example" # project directory
export BOINC_APP="example" # app name
export WU_TEMPLATE="templates/example_IN" # input file template
export RESULT_TEMPLATE="templates/example_OUT" # output file template
export RTE_LOCATION="$PROJECT_ROOT/Input/RTE.tar.gz" # RTEs, see below
```

The last variable is a tarball of runtime environments required by the job.
Known limitations

The BOINC back-end was designed around projects that use virtualisation. The prototype implementation in the current ARC version may not be generic enough to suit all BOINC projects.

When preparing a BOINC job, the ARC CE copies a tarball of the session directory to the BOINC project download area. Once the job is completed and the output uploaded to the BOINC project upload area, a modified assimilator daemon must be used to copy the result back to the session directory so that it can be retrieved by ARC clients or uploaded to Grid storage by the ARC CE.

ARC Information System

TBD

ARC CE Data Staging and Caching

RunTime Environments in ARC6

Understanding RunTime Environments

ARC Computing Element is a front-end to the various heterogeneous resource providers. To run jobs on the particular resource provider there are always set of software or workflow-specific paths, tools, libraries, environmental variables or even dynamic content that should be recreated in the job content.

To provide the flexible way of job runtime environment tuning ARC enforces the concept of the RunTime Environment (RTE).

ARC RunTime Environments (RTEs) provides the two functions:

Advertising indicate the available environment to be requested by end-users

Modifying job environment flexibly contextulalize job execution environment

Advertising RTEs

Advertising RTEs provide user interfaces to application software and other resources in a way that is independent of the details of the local installation of the application and computing platform (OS, hardware, etc.).

It addresses setups typically required by large research groups or user bases, dealing with a common set of software.

The actual implementation of particular RTE 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 (RTE) for all users.

Despite possibly different parameters or implementation, the same software addressed by the same RTE name known by community. It is also supported to add RTE versioning at the end of RTE name (after dash). RTE version will be used for resource matchmaking along with RTE name.

For example to request ENV/PROXY and APPS/HEP/ATLAS with version 20.1.0.1 or greater in xRSL job description:

```
(runTimeEnvironment="ENV/PROXY")
(runTimeEnvironment>="APPS/HEP/ATLAS-20.1.0.1")
```

It is always up to the local system administrators to take a decision whether to install and enable a particular RTE or not.

3.2. ARC Computing Element Deployment and Operations 133
Modifying job environment

The RTE content itself is a BASH script that aimed to run any arbitrary code during the job life cycle. The first argument of RTE script indicate the so-called RTE stage. If the job description specifies additional arguments for corresponding RTE those are appended starting at second position.

There are 3 stages of RTE execution:

**Stage 0** RTE script sourced before the creation of the job’s LRMS submission script. In this case the scripts are run by A-REX on the frontend (ARC CE), before the job is sent to the LRMS. Some environment variables are defined in this case, and can be changed to influence the job’s execution later. *TODO: list of grami attributes as a dedicated technical note*

**Stage 1** Embedded RTE function runs before the main job processing on the Worker Node under the LRMS. Such stage can prepare the environment for some third-party software package. The current directory in this case is the one which would be used for execution of the job. Variable `$HOME` also points to this directory.

**Stage 2** Embedded RTE function runs after the main job processing on the Worker Node under the LRMS. The main purpose is to clean possible changes done by *Stage 1* (like removing temporary files).

You can use *this template* to start writing custom RTE script that fulfill your needs.

**RunTime Environment script template**

You can start with the following template to write custom RTE script:
Operating RunTime Environments

In ARC6 release operating RunTime Environments is changed significantly and rely on ARC Control Tool:

![Diagram of Operating RunTime Environments](attachment:image.png)

---

3.2. ARC Computing Element Deployment and Operations
Installing RTE scripts

There are set of *System-defined RTEs* pre-installed with the ARC CE packages that aimed to fulfill common workflows.

ARC CE administrator can add additional RTE directories (so-called *User-defined RTEs*). This additional places should be specified in `arc.conf` using the `runtimedir` configuration option in `[arex]` block. Custom RTE scripts can be developed using *this template* as a starting point.

**Note:** In ARC6 directories with RTE script are local to ARC CE and SHOULD NOT be shared with worker nodes.

The RTE names used for *advertising* are implied by directories structure, e.g. in the `ENV/PROXY` the `ENV` is a directory inside *System RTEs* location and `PROXY` is a name of file.

Enabling RTEs

*Installed RTEs* should be enabled to be *advertised* and used during the job submission.

**By name**

To enable particular RTE by name run the following command:

```
arcctl rte enable ENV/PROXY
```

**By path**

Especially if you have several RTEs with the same name installed, you can choose the exact one by specifying filesystem path:

```
arcctl rte enable /usr/share/arc/rte/ENV/PROXY
```

**Using wildcards**

To enable several RTEs you can pass as many names as you want to the `arcctl` command. Additionally you can use `glob` (`man 7 glob`) wildcards in RTE names.

The following command will enable all `APPS/HEP/ATLAS` RTEs for SLC6 builds:

```
arcctl rte enable APPS/HEP/ATLAS-*-X86_64-SLC6-*
```

**Note:** It is advised to use wildcards to enable all user-defined RTEs during ARC5 to ARC6 upgrade.

**Dummy RTEs**

In case you need RTEs only for *advertising* but no need for script content, you can create *Dummy RTE* for specified name. The following command enables `APPS/MYAPP` RTE with empty content:

```
arcctl rte enable APPS/MYAPP --dummy
```
Default RTEs

Default RTEs aimed to address the workflows when advertising and implicit request in job description is not needed, however modification of every submitted job (adjusting memory, setting LRMS scratch, etc) is required on the resource provider.

Installed RTEs can be selected for default inclusion to job lifecycle with the following ARC Control Tool command:

```
arcctl rte default ENV/LRMS-SCRATCH
```

This will made ENV/LRMS-SCRATCH transparently added to each job and executed the same way as Enabled RTEs.

**Note:** You can use the same by-name, by-path and wildcard techniques as for enabling RTEs.

RTE Parameters

To achieve various cases due to heterogeneity of resource providers, some RTEs can be parametrized.

For example, the system-defined ENV/PROXY RTE that transfer the delegated proxy-certificate to the worker node can optionally transfer CA certificate directories. This optional part is controlled by COPY_CACERT_DIR parameter.

To check either RTE contains parameters and their default values, run the:

```
[root ~]# arcctl rte params-get ENV/PROXY
COPY_CACERT_DIR=Yes
```

You can also see the description and allowed values adding --long keyword.

To set RTE parameter value, the following command should be used:

```
arcctl rte params-set ENV/PROXY COPY_CACERT_DIR No
```

List available RTEs and their status

To view the summary of all installed, enabled and default RTEs run:

```
[root ~]# arcctl rte list
<output omitted>
APPS/HEP/ATLAS-20.8.0-X86_64-SLC6-GCC48-OPT (user, enabled)
APPS/HEP/ATLAS-20.8.1-X86_64-SLC6-GCC48-OPT (user, enabled)
APPS/HEP/ATLAS-20.8.2-X86_64-SLC6-GCC49-OPT (user, enabled)
<output omitted>
ENV/LRMS-SCRATCH (system, default)
ENV/PROXY (system, masked, disabled)
ENV/PROXY (user, enabled)
ENV/RTE (system, disabled)
ENV/RUNTIME/ALIEN-2.17 (user, enabled)
VO-biomed-CVMFS (dummy, enabled)
```

The first tag describe RTE origin (system, user or dummy). The following tags shows the status.

The special masked keyword indicates that RTE name used more that once and by-name operations will apply to other RTE script. In example ENV/PROXY will be enabled from user-defined location and system-defined will be masked. However it is possible to enable masked RTE by path.
Listing the particular kind of RTEs (e.g. enabled) is possible with appropriate argument (see ARC Control Tool for all available options):

```
[root ~]# arcctl rte list --enabled
<output omitted>
APPS/HEP/ATLAS-20.8.2-X86_64-SLC6-GCC49-OPT
ENV/PROXY
ENV/RUNTIME/ALIEN-2.17
VO-biomed-CVMFS
```

The long listing allows to get the detailed pointers to RTEs locations and descriptions:

```
[root ~]# arcctl rte list --long
System pre-defined RTEs in /usr/share/arc/rte:
   ENV/PROXY # copy proxy certificate to the job session
   ENV/RTE # copy RunTimeEnvironment scripts to the job session
   ENV/LRMS-SCRATCH # enables the usage of local to WN scratch session
User-defined RTEs in /etc/arc/rte:
   ENV/RUNTIME/ALIEN-2.17 # RTE Description is Not Available
   ENV/PROXY # RTE Description is Not Available
Enabled RTEs:
   ENV/RUNTIME/ALIEN-2.17 -> /etc/arc/rte/ENV/RUNTIME/ALIEN-2.17
   ENV/PROXY -> /etc/arc/rte/ENV/PROXY
Default RTEs:
   ENV/LRMS-SCRATCH -> /usr/share/arc/rte/ENV/LRMS-SCRATCH
```

View RTE content

Dumping the content of RTE that will be embedded to job script is possible with cat action:

```
[root ~]# arcctl rte cat ENV/LRMS-SCRATCH
SCRATCH_VAR="LOCALTMP"
# description: enables the usage of local to WN scratch directory defined by LRMS
# param:SCRATCH_VAR:string:WORKDIR:Variable name that holds the path to job-
# specific WN scratch directory
SCRATCH_VAR="${SCRATCH_VAR:-WORKDIR}"
if [ "x$1" = "x0" ]; then
   RUNTIME_LOCAL_SCRATCH_DIR="\${${SCRATCH_VAR}}"
fi
```

Disable and Undefault RTEs

Enabled RTEs can be disabled running:

```
arcctl rte disable ENV/PROXY
```

The similar operation for default RTEs is called undefault:

```
arctt1 rte undefault ENV/LRMS-SCRATCH
```

Note: You can use the same by-name, by-path and wildcard techniques as for enabling
System-defined RunTime Environments shipped with ARC

**ENV/PROXY**

Export delegated credentials (proxy certificate) to the job’s session directory. Optionally copies CA certificates directory from ARC CE to session directory.

Sets the `X509_USER_PROXY`, `X509_USER_CERT` and `X509_CERT_DIR` to make files instantly available to client tools.

**Parameters:**

- `COPY_CACERT_DIR = Yes/No` - If set to Yes, CA certificate directory will be copied to the session directory along with proxy certificate. Default is Yes.
- `USE_DELEGATION_DB = Yes/No` - If set to Yes RTE will try to extract proxy certificate from A-REX delegation DB (works in limited number of cases). Default is No.

**ENV/RTE**

Copy RunTime Environment scripts to the job session directory for some workloads that require files itself instead of embedding the RTE to jobsctipt.

Designed to be used as default RTE.

Has no parameters.

**ENV/LRMS-SCRATCH**

Many resource providers uses `scratchdir` to move files to local to worker node disk before running the job.

There is a useful case when local scratch is created dynamically by LRMS (e.g. in the job prologue) and than cleaned up automatically after the job completion. The scratch place should be indicated by some environmental variable that holds a path to such LRMS-defined scratch directory.

This RTE designed to be used as default RTE to enable this optional functionality.

**Parameters:**

- `SCRATCH_VAR = name` - Variable name that holds the path to job-specific WN scratch directory. Default is `WORKDIR`.
- `TMPDIR_LOCATION = path` - Define the `TMPDIR` path on WN. Variable names can be used as a part of the path, e.g. `$WORKDIR/tmp`.

**ENV/CONDOR/DOCKER**

ARC HTCondor backend supports submission to Docker universe. This RTE enables this feature on-demand.

RTE can be use by end-users when enabled. The RTE argument defines the Docker image name to be used, e.g:

```plaintext
(runtimeenvironment="ENV/DOCKER" "debian")
```

RTE can also be used as default RTE to enforce Docker universe submission for any job. Docker image should be set with RTE parameter.

**Parameters:**

- `DOCKER_IMAGE = name` - Docker image to use for submitted jobs by default.
ENV/CANDYPOND (experimental)

Makes available the arccandypond tool for usage inside the job script on the Worker Nodes (including necessary environmental variables for it’s operation).

Note: The CandyPond service itself should be enabled (defining the [arex/ws/candypond] block) on ARC CE as well.

Parameters:

- CANDYPOND_URL = url - Redefine the URL of CandyPond service (default is auto – ARC CE URL used for job submission will be used automatically)

Measuring accounting metrics of the job

ARC CE has built-in capabilities to collect information about per-job resource consumption. This includes both ARC CE resources (e.g. data transfers, software environments) and worker nodes resources (e.g. CPU and memory usage). The full list of attributes stored in the A-REX Accounting Records (AAR) can be found in the JURA Accounting Technical Details. A-REX can use different methods (described below) to measure memory and CPU usage on the worked nodes, depending on their availability in the particular deployment case.

Measuring memory and CPU usage on the WN with cgroups

New in version 6.2.

When recent versions of GNU/Linux OS are used on the worker nodes the most precise and transparent way to measure all job workload is to rely on cgroups kernel subsystem. Any systemd-based Linux distribution relies on cgroups heavily and they are already used.

Note: Some older operating systems may require to mount cgroups tree explicitly. For example in RHEL6 it can be easily done with libcgroup:

```
[root ~]# yum install libcgroup
[root ~]# service cgconfig start
```

The benefit of using cgroups is that everything will be accounted. Even if several payloads are executed (e.g. in pilot mode) or extra helper processes are spawned - the resource accounting will be accurate for the all workload done.

Enabling cgroups usage

To be able to use cgroups for accounting ARC needs an extra tool installed on the worker nodes – the arc-job-cgroup. Based on the tool availability job script will or will NOT use cgroups for measuring accounting metrics automatically.

The arc-job-cgroup tool is available for majority of OSes as a packaged binary build as a part of ARC distribution (nordugrid-arc-wn package).

If it is not possible to install the packaged version for some reason, it is easy to compile the source code that is only 300 line of pure C code with standard C library calls only.
How ARC operates cgroups

The idea behind LRMS-independent cgroup-based resource usage measurements in ARC is to:

• create a child cgroups for memory and cpuacct controllers
• put the jobscript process into created cgroups (this will automatically catch all child processes)
• collect the accounting data at the end of jobscript
• remove the child cgroup created at the beginning (moving all processes to parent cgroup)

If cgroups are used in Kernel, the process already belongs to some cgroup. It can be either a root cgroup (used for all processes) or some dedicated cgroup created by LRMS with cgroups support, container management system, etc.

All resources used by child cgroup are accounted in parent cgroup. Moreover all parent-defined limits are inherited and enforced as well. So creating another child cgroup in hierarchy is safe from all points of view.

Warning: Creating a child cgroup and put a task into it requires root privileges. This is the reason behind the SUID bit for arc-job-cgroup. However the code itself is as simple as the mkdir. You can review this 300 lines to reason out of possible fears.

If arc-job-cgroup tool is not available, cgroups tree is not mounted or any other issues with cgroups creation, the job script code fall back to the GNU time measurement method.

Measuring memory and CPU usage on the WN with GNU time

The GNU time utility is capable of measuring and displaying information about the resources used by the executable it runs.

It is used as a part of ARC-generated job script if found on the worker node.

Note: Changed in version 6.2.

In case of successful cgroups usage, GNU time will NOT be used by job script.

Warning: The GNU time is a separate binary typically installed by dedicated package. Do not mix it up with built-in version of time in your shell (e.g. bash, zsh).

Typically you can install it with e.g. yum install time or similar package management command.

For non-standard location of GNU Time the gnu_time configuration option can be used to define it.

If the GNU time utility is not available the job will run as it is and only LRMS-provided metrics will be accounted.
Using LRMS-provided metrics

After the job had finished execution in LRMS, the batch system backend scan-script extract accounting information about the job from LRMS, either executing command line clients, parsing logs or using API.

The exact data measurements and the method of these data collection is completely depends on LRMS backend implementation and differs from one backed to another.

The common metrics includes LRMSStartTime and LRMSEndTime. There are also typically some memory and CPU usage metrics available.

Accounting with JURA

The Job Usage Reporter of ARC (JURA) is a component which is capable to create standard-compliant usage records from job usage information provided by the A-REX (Job Log files) and send the records to remote accounting services.

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

- Usage Record 1.0 (UR) XML format to be sent to an SGAS LUTS (Logging and Usage Tracking Service)
- Compute Accounting Record (CAR) XML format to be sent to APEL

Overview

Figure shows the overview of accounting workflow in ARC. More details can be found in JURA Accounting Technical Details.

![ARC Computing Element](image)

Fig. 3.3: Accounting records processing in ARC CE: creating, processing, publishing and archiving

When accounting is enabled the sequence of events is the following:

1. A-REX writes the per-job accounting records to Job Logs directory (logs directory inside the controldir).
2. A-REX periodically runs jura. Default and minimum period is hourly but can be increased with urdelivery_frequency option.
3. Jura reads jobs accounting information from Job Logs and depending on the targets configured in arc.conf creates an accounting records according to required format.
4. Jura publish accounting records to configured accounting services.

5. If records archiving is enabled, JURA also put the records to Accounting Archive.

6. When archiving functionality is used A-REX also runs the jura-archive-manager process that is managing archive structure and indexing the records in the local database. Local accounting database then can be used to lookup statistics or republish data.

Enabling accounting records reporting

JURA is a stand-alone binary application that is part of the A-REX package and can be enabled with [arex/jura] block.

JURA has dedicated log file defined by logfile option. Log rotation has been set for default /var/log/arc/jura.log location. Accounting logs can also be viewed with arcctl:

```
[root ~]# arcctl accounting logs
```

Accounting services for sending the records are configured with dedicated sub-blocks. You need to define a separate block with an unique targetname for every target server used.

The usage record of each job will be reported to all of the destinations, unless vofiler option configured for some of them to filter records by VO name.

Configuring reporting to SGAS

The SGAS sub-block enables and configures an SGAS accounting server as a target destination to which JURA will send properly formatted Usage Record 1.0 (UR) XML records.

The targeturl option is the only mandatory parameter to configure SGAS target. In the specific setup cases you can also apply VO filtering and set prefix for local job IDs.

Example:

```
[arex/jura/sgas: NeIC]
targeturl = https://grid.uio.no:8001/logger
urbatchsize = 80
```

Configuring reporting to APEL

The APEL sub-block enables and configures an APEL accounting server as a target destination to which JURA will send properly formatted Compute Accounting Record (CAR) XML records.

The targeturl option defines the APEL broker URL to send records to. Unlike APEL native client that relies on Top-BDII infrastructure in runtime to find the blockers, ARC uses dedicated target URL in configuration. However it is possible to lookup available targets with arcctl:

```
[root ~]# arcctl accounting apel-brokers
http://mq.cro-ngi.hr:6163/
http://broker-prod1.argo.grnet.gr:6163/
[root ~]# arcctl accounting apel-brokers --ssl
https://mq.cro-ngi.hr:6162/
https://broker-prod1.argo.grnet.gr:6162/
```

SSL or non-SSL connection should be defined accordingly with use_ssl option.

You also need GOCDB name of the resource and APEL topic to which JURA will publish the accounting records. For correct production accounting setup it is recommended to specify resource benchmarking results.

Example:
NorduGrid ARC 6 Information, Release ARC6

Note: Publishing to APEL relies on SSM framework that has own logfile viewable with `arcctl accounting logs --ssm`.

Records archiving

The archiving functionality allows to store generated usage records in a specified directory on the disk. The `[arex/jura/archiving]` block enables this functionality. Directory path for the jura archived usage records is defined with `archivedir` option.

Once archive is enabled JURA wrote generated usage records to files named `usagerecord[CAR].<jobid>.<random>`. If a A-REX 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 content of the archive file is used without change (the creation time stamp is also retained).

Note: Records archive can grow very rapidly on heavy loaded sites with huge amount of jobs coming daily. To limit the archiving time use `archivettl` option.

To better organize archived records and index the accounting data in the local database the `jura-archive-manager` helper process is run periodically by A-REX when archive is enabled.

The archive manager:

- read the resource usage data and store it to indexed SQLite database file `accounting.db`.
- move files according to the following directory structure: `YYYY-MM/DD/<jj>/<kkkkk>.<random>.{UR|CAR}`, where `jj` is first 2 characters of `<jobid>` and `kkkkk` is the rest of `<jobid>`

Lookup local accounting data

Data in local accounting database can be viewed with `arcctl` for each type of destination. Timeframe of interest and many other filters can be specified, e.g.:

```
[root ~]# arcctl accounting stats -t apel -b 2018-11-01 -e 2018-11-30
Statistics for APEL jobs from 2018-11-01 00:00:02 till 2018-11-29 23:59:13:
  Number of jobs: 355168
  Total WallTime: 48146 days, 18:06:50
  Total CPUtilme: 71927 days, 10:33:33
[root ~]# arcctl accounting stats -t apel -v ops belle atlas
ops.ndgf.org
[root ~]# arcctl accounting stats -t apel --filter-vo belle
Statistics for APEL jobs from 2015-04-13 07:28:52 till 2019-04-02 00:25:57:
  Number of jobs: 869400
  Total WallTime: 216866 days, 3:33:33
  Total CPUtilme: 193604 days, 19:40:58
```
Republishing records

When something goes wrong with accounting services, network, etc there is possible need of republishing local records again.

**Warning:** Republishing is ONLY possible when Records archiving is enabled and records are still exists for required time period.

The ARC Control Tool should be used to republish data. It is tightly integrated with accounting archive database: arcct1 will lookup needed records in database for defined republishing period, prepare them for republishing and than invokes jura with correct configuration settings.

**Warning:** It is strongly recommended to AVIOD running republishing in ARC5-way, executing the jura command manually. Even if you managed to get it work it can cause unpredicted results.

Define the timeframe of interest and run the command to republish data:

```
```

Job scratch area

ARC allows to configure different approaches to manage the job scratch area during the job life cycle.

The key elements are:

- **job session directory** directory on the ARC CE where the job files (coming from the client and data-staging framework) are located.
- **job scratch directory** directory on LRMS-managed worker nodes (WNs) where all I/O during computation is performed.

*Job session directory* is configured with `sessiondir` configuration option. It is possible to configure several session directories. A-REX than will select one of available directories and append the Job ID to the path.

There are several configuration options that affect the selection of **job scratch directory**:

- **shared filesystem** defines is the **job session directory** is shared between ARC CE and WNs (by means of e.g. NFS). Sets the environment variable `RUNTIME_NODE_SEES_FRONTEND`.
- **scratchdir** defines the path to **job scratch directory** on the WN. Sets the environment variable `RUNTIME_LOCAL_SCRATCH_DIR`.
- **shared scratch** defines that the WNs scratch directory can be accessible from ARC CE (by means of e.g. NFS) using the configured path. Sets the environment variable `RUNTIME_FRONTEND_SEES_NODE`.

**Note:** Described environmental variables can be redefined by RunTime Environments dynamically. For example `ENV/LRMS-SCRATCH` can be used to utilize local scratch that created dynamically by LRMS.

Compute inside shared session directory

In the simplest case, where **job session directory** is shared between ARC CE and WNs (e.g. via NFS) and accessible on the same path - the **job scratch directory** is the **job session directory**.

Configuration:
Fig. 3.4: Sessiondir is shared between ARC CE and WNs. No local scratchdir defined.

```
[arex]
shared_filesystem = yes
```

**Compute inside local WN scratch directory**

**Session directory is shared**

For I/O performance reasons it is possible to made computations inside local to WN disk area.

In this case job scratch directory is created in the configured local directory and files from the job session directory are moved to job scratch before execution starts.

Only files representing the job’s stdout and stderr are placed in the original job session directory and soft-linked in scratch.

**Configuration:**

```
[arex]
shared_filesystem = yes
scratchdir = /mnt/scratch/arc
```

**Session directory is NOT shared**

If the session directory is not shared the data movement between ARC CE and WN is done by means of LRMS backend. How to implement the data movement between the ARC CE and WN depends on the particular batch system backend used. This is not described here, please refer to your batch system manuals.

**Warning:** If job session directory is not shared (shared_filesystem = no) job scratch directory SHOULD BE defined (scratchdir = path) to instruct LRMS where to stage-in the files. Otherwise job submission to LRMS will fail.

After all input files are gathered in the job session directory on ARC CE, the LRMS copies files to the job scratch directory on WN (Figure a)).

The job performs all I/O using local job scratch directory. After execution all declared output files (including stdout and stderr) are staged out to the job session directory on ARC CE by LRMS (Figure b)).

---

¹ For example in the PBS this corresponds to `#PBS -W stagein` options.
Fig. 3.5: Session dir is not shared between ARC CE and WNs.
When the output job files are available in the job session directory on ARC CE they are ready to be uploaded to external storage elements or be downloaded by the user (Figure c)).

```bash
[arcx]
shared_filesystem = no
scratchdir = /mnt/scratch/arc
```

### Scratch directory is readable from ARC CE

Another advanced use-case covers the case when the job scratch directory is readable from ARC CE.

![Diagram](image)

**Fig. 3.6: Sessiondir is not shared but scratch directory is exported to ARC CE**

In this case, after the movement of files from job session directory on ARC CE to job scratch directory on WN (Figure a)) the original job session directory is removed and replaced with a soft-link to a job scratch directory as seen on the frontend.

The job performs all I/O using local job scratch directory. All files are also directly available on the frontend through a soft-link.

After execution, the soft-link is replaced with the directory and all files from job scratch directory are moved back.
to the original job session directory (Figure b).
All files in the job session directory on ARC CE are ready to be staged out (Figure c).

```
[arex]
scratchdir = /mnt/scratch/arc
shared_scratch = /net/wns/scratch/arc
```

### 3.2.6 ARC Admin Tools Reference

#### ARC Control Tool

NorduGrid ARC Computing Element Control Tool

```
usage: arcctl [-h] [-c CONFIG] [-d {CRITICAL,ERROR,WARNING,INFO,DEBUG}] COMPONENT ...
```

**Named Arguments**

- `-c, --config` config file location (default is /etc/arc.conf)
- `-d, --debug` Possible choices: CRITICAL, ERROR, WARNING, INFO, DEBUG
  
  verbosity level (default is “WARNING”)
  Default: “WARNING”

**ARC CE Components**

**COMPONENT** Possible choices: rte, job, service, deploy, accounting, config, test-ca, cache

**DESCRIPTION**

**Sub-commands:**

**rte**

RunTime Environments

```
arcctl rte [-h] ACTION ...
```

**RunTime Environments Actions**

**ACTION** Possible choices: enable, disable, list, default, undefault, cat, params-get, params-set, params-unset

**DESCRIPTION**

**Sub-commands:**

**enable**

Enable RTE to be used by A-REX
arcctl rte enable [-h] [-f] [-d] rte [rte ...]

Positional Arguments

rte
RTE name

Named Arguments

-f, --force
Force RTE enabling
Default: False

-d, --dummy
Enable dummy RTE that do nothing but published in the infosys
Default: False

disable
Disable RTE to be used by A-REX

arcctl rte disable [-h] rte [rte ...]

Positional Arguments

rte
RTE name

list
List RunTime Environments

arcctl rte list [-h] [-l] [-e | -d | -a | -s | -u | -n]

Named Arguments

-l, --long
Detailed listing of RTEs
Default: False

-e, --enabled
List enabled RTEs
Default: False

-d, --default
List default RTEs
Default: False

-a, --available
List available RTEs
Default: False

-s, --system
List available system RTEs
Default: False

-u, --user
List available user-defined RTEs
Default: False
-n, --dummy

List dummy enabled RTEs

Default: False

default

Transparently use RTE for every A-REX job

```
arcctl rte default [-h] [-f] rte [rte ...]
```

**Positional Arguments**

- **rte**
  - RTE name

**Named Arguments**

- **-f, --force**
  - Force RTE enabling
  - Default: False

undefault

Remove RTE from transparent A-REX usage

```
arcctl rte undefault [-h] rte [rte ...]
```

**Positional Arguments**

- **rte**
  - RTE name

cat

Print the content of RTE file

```
arcctl rte cat [-h] rte
```

**Positional Arguments**

- **rte**
  - RTE name

params-get

List configurable RTE parameters

```
arcctl rte params-get [-h] [-l] rte
```

**Positional Arguments**

- **rte**
  - RTE name
Named Arguments

- **-l, --long**  
  Detailed listing of parameters  
  Default: False

**params-set**

Set configurable RTE parameter

```
arcctl rte params-set [-h] rte parameter value
```

Positional Arguments

- **rte**  
  RTE name
- **parameter**  
  RTE parameter to configure
- **value**  
  RTE parameter value to set

**params-unset**

Use default value for RTE parameter

```
arcctl rte params-unset [-h] rte parameter
```

Positional Arguments

- **rte**  
  RTE name
- **parameter**  
  RTE parameter to unset

**job**

A-REX Jobs

```
arcctl job [-h] [-t CACHETTL] ACTION ...
```

Named Arguments

- **-t, --cachettl**  
  GM-Jobs output caching validity in seconds (default is 30)  
  Default: 30

**Jobs Control Actions**

- **ACTION**  
  Possible choices: list, script, log, info, stdout, stderr, attr, kill, killall, clean, cleanall, stats
  DESCRIPTION
Sub-commands:

list

List available A-REX jobs

```
arcctl job list [-h] [-l] [-s {ACCEPTED, PREPARING, SUBMIT, INLRMS, FINISHING, FINISHED, DELETED, CANCELING}] [-o OWNER]
```

Named Arguments

- **-l, --long** Detailed listing of jobs
  
  Default: False
- **-s, --state** Possible choices: ACCEPTED, PREPARING, SUBMIT, INLRMS, FINISHING, FINISHED, DELETED, CANCELING
  
  Filter jobs by state
- **-o, --owner** Filter jobs by owner

script

Display job script submitted to LRMS

```
arcctl job script [-h] jobid
```

Positional Arguments

- **jobid** Job ID

log

Display job log

```
arcctl job log [-h] [-f] [-s] jobid
```

Positional Arguments

- **jobid** Job ID

Named Arguments

- **-f, --follow** Follow the job log output
  
  Default: False
- **-s, --service** Show ARC CE logs containing the jobID instead of job log
  
  Default: False
info

Show job main info

```
arcctl job info [-h] jobid
```

Positional Arguments

- `jobid`: Job ID

stdout

Show job executable stdout

```
arcctl job stdout [-h] [-f] jobid
```

Positional Arguments

- `jobid`: Job ID

Named Arguments

- `-f, --follow`: Follow the job log output
  
  Default: False

stderr

Show job executable stderr

```
arcctl job stderr [-h] [-f] jobid
```

Positional Arguments

- `jobid`: Job ID

Named Arguments

- `-f, --follow`: Follow the job log output
  
  Default: False

attr

Get

```
arcctl job attr [-h] jobid [attr]
```
Positional Arguments

jobid  Job ID
attr   Attribute name

kill

Cancel job
	arcctl job kill [-h] jobid [jobid ...]

Positional Arguments

jobid  Job ID

killall

Cancel all jobs
	arcctl job killall [-h] 
		[-s {ACCEPTED,PREPARING,SUBMIT,INLRMS,FINISHING,FINISHED, 
		DELETE,CANCELING}] 
		[-o OWNER]

Named Arguments

-s, --state  Possible choices: ACCEPTED, PREPARING, SUBMIT, INLRMS, FINISHING, FINISHED, DELETED, CANCELING
            Filter jobs by state
-o, --owner  Filter jobs by owner

clean

Clean job
	arcctl job clean [-h] jobid [jobid ...]

Positional Arguments

jobid  Job ID

cleanall

Clean all jobs
	arcctl job cleanall [-h]
		[-s {ACCEPTED,PREPARING,SUBMIT,INLRMS,FINISHING,FINISHED, 
		DELETE,CANCELING}] 
		[-o OWNER]
Named Arguments

- **-s, --state**
  Possible choices: ACCEPTED, PREPARING, SUBMIT, INLRMS, FINISHING, FINISHED, DELETED, CANCELING
  Filter jobs by state

- **-o, --owner**
  Filter jobs by owner

stats

Show jobs statistics

```
arcctl job stats [-h] [-l] [-t | -d]
```

Named Arguments

- **-l, --long**
  Detailed output of stats
  Default: False

- **-t, --total**
  Show server total stats
  Default: False

- **-d, --data-staging**
  Show server datastaging stats
  Default: False

service

ARC CE services control

```
arcctl service [-h] ACTION ...
```

Services Actions

**ACTION**

Possible choices: enable, disable, start, restart, stop, list

DESCRIPTION

Sub-commands:

enable

Enable ARC CE services

```
arcctl service enable [-h] [--now] (-a | -s SERVICE)
```

Named Arguments

- **--now**
  Start the services just after enable
  Default: False
-a, --as-configured  Use information from arc.conf to get services list  
Default: False
-s, --service  Service name

**disable**

Disable ARC CE services

```bash
arcctl service disable [-h] [--now] (-a | -s SERVICE)
```

**Named Arguments**

- --now  Stop the services just after disable  
Default: False
- -a, --as-configured  Use information from arc.conf to get services list  
Default: False
- -s, --service  Service name

**start**

Start ARC CE services

```bash
arcctl service start [-h] (-a | -s SERVICE)
```

**Named Arguments**

- -a, --as-configured  Use information from arc.conf to get services list  
Default: False
- -s, --service  Service name

**restart**

Restart ARC CE services

```bash
arcctl service restart [-h] (-a | -s SERVICE)
```

**Named Arguments**

- -a, --as-configured  Use information from arc.conf to get services list  
Default: False
- -s, --service  Service name
stop

Stop ARC CE services

```
arcctl service stop [-h] (-a | -s SERVICE)
```

**Named Arguments**

- `-a`, `--as-configured`  Use information from arc.conf to get services list
  Default: False
- `-s`, `--service`  Service name

list

List ARC CE services and their states

```
arcctl service list [-h] [-i | -e | -a]
```

**Named Arguments**

- `-i`, `--installed`  Show only installed services
  Default: False
- `-e`, `--enabled`  Show only enabled services
  Default: False
- `-a`, `--active`  Show only running services
  Default: False

deploy

Third party components deployment

```
arcctl deploy [-h] ACTION ...
```

**Deployment Actions**

```
ACTION  Possible choices: voms-lsc, igtf-ca, iptables-config
DESCRIPTION
```

**Sub-commands:**

**voms-lsc**

Deploy VOMS list-of-certificates files

```
arcctl deploy voms-lsc [-h] (-v VOMS | -e) [--pythonssl] vo
```
Positional Arguments

vo VO Name

Named Arguments

-v, --voms VOMS-Admin URL
-e, --egi-vo Fetch information from EGI VO database
Default: False
--pythonssl Use Python SSL module to establish TLS connection (default is to call external OpenSSL binary)
Default: False

igtf-ca

Deploy IGTF CA certificates

```
arcctl deploy igtf-ca [-h] [-i {igtf, egi-trustanchors, nordugrid}] {classic, iota, mics, slcs} [{classic, iota, mics, slcs} ...]
```

Positional Arguments

bundle Possible choices: classic, iota, mics, slcs
IGTF CA bundle name

Named Arguments

-i, --installrepo Possible choices: igtf, egi-trustanchors, nordugrid
Add specified repository that contains IGTF CA certificates

iptables-config

Generate iptables config to allow ARC CE configured services

```
arcctl deploy iptables-config [-h] [--any-state] [--multiport]
```

Named Arguments

--any-state Do not add ‘–state NEW’ to filter configuration
Default: False
--multiport Use one-line multiport filter instead of per-service entries
Default: False
accounting

Accounting records management

```
arcctl accounting [-h] ACTION ...
```

## Accounting Actions

### ACTION

Possible choices: republish, logs, stats, apel-brokers

### DESCRIPTION

### Sub-commands:

**republish**

Republish archived usage records

```
arcctl accounting republish [-h] -b START_FROM -e END_TILL
   (-a APEL_URL | -s SGAS_URL) [--db-init]
   [-t {/queue/global.accounting.cpu.central,/queue/
   →global.accounting.test.cpu.central}]
```

**Named Arguments**

- `-b`, `--start-from`  
  Limit the start time of the records (YYYY-MM-DD [HH:mm:ss])

- `-e`, `--end-till`  
  Limit the end time of the records (YYYY-MM-DD [HH:mm:ss])

- `-a`, `--apel-url`  
  Specify APEL server URL (e.g. https://mq.cro-ngi.hr:6163)

- `-s`, `--sgas-url`  
  Specify APEL server URL (e.g. https://grid.uio.no:8001/logger)

- `--db-init`  
  Force accounting database init from arcctl

  Default: False

- `-t`, `--apel-topic`  
  Possible choices: /queue/global.accounting.cpu.central, /queue/global.accounting.test.cpu.central

  Redefine APEL topic (default is “/queue/global.accounting.cpu.central”)  

  Default: “/queue/global.accounting.cpu.central”

**logs**

Show accounting logs

```
arcctl accounting logs [-h] [-s]
```

**Named Arguments**

- `-s`, `--ssm`  
  Show SSM logs instead of Jura logs

  Default: False
stats

Show archived records statistics

```
arcctl accounting stats [-h] -t {apel,sgas} [-b START_FROM] [-e END_TILL]
  [--db-init] [--filter-vo FILTER_VO]
  [--filter-user FILTER_USER] [-j | -w | -c | -v | -u]
```

Named Arguments

- `-t`, `--type` Possible choices: apel, sgas
  Accounting system type
- `-b`, `--start-from` Limit the start time of the records (YYYY-MM-DD [HH:mm:ss])
- `-e`, `--end-till` Limit the end time of the records (YYYY-MM-DD [HH:mm:ss])
- `--db-init` Force accounting database init from arcctl
  Default: False
- `--filter-vo` Count only the jobs owned by this VO(s)
- `--filter-user` Count only the jobs owned by this user(s)
- `-j`, `--jobs` Show number of jobs
  Default: False
- `--walltime` Show total WallTime
  Default: False
- `--cputime` Show total CPUTime
  Default: False
- `--vos` Show VOs that owns jobs
  Default: False
- `--users` Show users that owns jobs
  Default: False

apel-brokers

Fetch available APEL brokers from GLUE2 Top-BDII

```
arcctl accounting apel-brokers [-h] [-t TOP_BDII] [-s]
```

Named Arguments

- `-t`, `--top-bdii` Top-BDII LDAP URI (default is “ldap://lcg-bdii.cern.ch:2170”)
- `--ssl` Query for SSL brokers
  Default: False
config

ARC CE configuration control

```
arcctl config [-h] ACTION ...
```

Config Actions

```
ACTION Possible choices: dump, get, describe, brief, verify
DESCRIPTION
```

Sub-commands:

`dump`

Dump ARC CE running configuration

```
arcctl config dump [-h]
```

`get`

Print configuration option value

```
arcctl config get [-h] block option
```

Positional Arguments

```
  block Name of configuration block (without square breakets)
  option Configuration option name
```

`describe`

Describe configuration option

```
arcctl config describe [-h] [-r REFERENCE] block option
```

Positional Arguments

```
  block Name of configuration block (without square breakets)
  option Configuration option name
```

Named Arguments

```
  -r, --reference Redefine arc.conf.reference location (default is “/usr/share/doc/nordugrid-arc//arc.conf.reference”)  
                  Default: “/usr/share/doc/nordugrid-arc//arc.conf.reference”
```
brief

Print configuration brief points

```
arcctl config brief [-h] [-t {storage,logs}]
```

Named Arguments

- `-t, --type`  Possible choices: storage, logs
  Show brief only for provided options type

verify

Verify ARC CE configuration syntax

```
arcctl config verify [-h] [-v {quiet,verbose,debug,skip-warnings}]
```

Named Arguments

- `-v, --verbosity`  Possible choices: quiet, verbose, debug, skip-warnings
  Controls verbosity of config validation output

test-ca

ARC Test CA control

```
arcctl test-ca [-h] [--ca-id CA_ID] ACTION ...
```

Named Arguments

- `--ca-id`  Define CA ID to work with (default is to use hostname-based hash)

Test CA Actions

<table>
<thead>
<tr>
<th>ACTION</th>
<th>Possible choices: init, cleanup, hostcert, usercert</th>
</tr>
</thead>
</table>

Sub-commands:

init

Generate self-signed TestCA files

```
arcctl test-ca init [-h]
[-d {md2,md4,md5,mdc2,sha1,sha224,sha256,sha384,sha512}]
[-v VALIDITY] [-f]
```
Named Arguments

-d, --digest
Possible choices: md2, md4, md5, mdc2, sha1, sha224, sha256, sha384, sha512
Digest to use (default is “sha256”)
Default: “sha256”

-v, --validity
Validity of certificate in days (default is 30)
Default: 30

-f, --force
Overwrite files if exist
Default: False

cleanup

Cleanup TestCA files

```
arcctl test-ca cleanup [-h]
```

hostcert

Generate and sign testing host certificate

```
arcctl test-ca hostcert [-h]
[ -d {md2,md4,md5,mdc2,sha1,sha224,sha256,sha384,sha512}] [-v VALIDITY] [-n HOSTNAME] [-f] [-t]
```

Named Arguments

-d, --digest
Possible choices: md2, md4, md5, mdc2, sha1, sha224, sha256, sha384, sha512
Digest to use (default is “sha256”)
Default: “sha256”

-v, --validity
Validity of certificate in days (default is 30)
Default: 30

-n, --hostname
Generate certificate for specified hostname instead of this host

-f, --force
Overwrite files if exist
Default: False

-t, --export-tar
Export tar archive to use from another host
Default: False

usercert

Generate and sign testing user certificate
arcctl test-ca usercert [-h]
   [-d {md2,md4,md5,mdc2,sha1,sha224,sha256,sha384,sha512}]
   [-v VALIDITY] [-n USERNAME] [-i INSTALL_USER] [-t]
   [-f] [--no-auth]

Named Arguments

- **-d, --digest**
  Possible choices: md2, md4, md5, mdc2, sha1, sha224, sha256, sha384, sha512
  Digest to use (default is “sha256”)
  Default: “sha256”

- **-v, --validity**
  Validity of certificate in days (default is 30)
  Default: 30

- **-n, --username**
  Use specified username instead of automatically generated

- **-i, --install-user**
  Install certificates to $HOME/.globus for specified user instead of workdir

- **-t, --export-tar**
  Export tar archive to use from another host
  Default: False

- **-f, --force**
  Overwrite files if exist
  Default: False

- **--no-auth**
  Do not add user subject to allowed list
  Default: False

cache

ARC A-REX Cache control

arcctl cache [-h] ACTION ...

A-REX Cache Actions

ACTION Possible choices: stats, list, is_cached

DESCRIPTION

Sub-commands:

stats

Show cache usage statistics

arcctl cache stats [-h]
NorduGrid ARC 6 Information, Release ARC6

list

List cached URLs

```
arcctl cache list [-h] [-l]
```

Named Arguments

- **-l, --long**
  Output paths to cached files
  Default: False

is-cached

Checks is the URL already in A-REX cache

```
arcctl cache is-cached [-h] [-q] url
```

Positional Arguments

- **url**
  URL to check

Named Arguments

- **-q, --quiet**
  Do not output path to cached file
  Default: False

ARC Configuration Parser

Nordugrid ARC configuration parser

```
```

Named Arguments

- **--debug**
  Possible choices: CRITICAL, ERROR, WARNING, INFO, DEBUG
  verbosity level (default is “WARNING”)
  Default: “WARNING”

Runtime configuration

Work with runtime configuration that includes default values

- **--load**
  load ARC runtime configuration
  Default: False
--save  save ARC runtime configuration
          Default: False
-r, --runconfig  runtime config file location (default is “/var/run/arc/arc.runtime.conf”)
          Default: “/var/run/arc/arc.runtime.conf”

Configuration files

Initial ARC configuration files
-c, --config  config file location (default is “/etc/arc.conf”)
          Default: “/etc/arc.conf”
-d, --defaults  defaults file location (default is “/usr/share/arc/arc.parser.defaults”)
          Default: “/usr/share/arc/arc.parser.defaults”

Getting values

Get blocks and configuration option values
-b, --block  block name (can be specified several times)
-o, --option  option name
-s, --subblocks  match subblocks against supplied block name(s)
          Default: False
-e, --export  Possible choices: bash, json
          export configuration to the defined format
-f, --export-filter  limit bash export to specified options only

ARCHERY Manage Tool

NorduGrid ARCHERY management tool

usage: archery-manage [-h] [-d {CRITICAL,ERROR,WARNING,INFO,DEBUG}] -s SOURCE
                      [-o {services,zonefile,object-debug,endpoints,arc-CEs}]
                      [--json] [--output-all] [-u] [--domain DOMAIN]
                      [--dns-master-ip DDNS_MASTER_IP]
                      [--dns-tsig-keyfile DDNS_TSIG_KEYFILE]
                      [--dns-tsig-algorithm \{HMAC-SHA512,HMAC-SHA256,HMAC-MD5,
                        HMAC-SHA1,HMAC-SHA384,HMAC-SHA224\}]
                      [--ttl TTL] [--threads THREADS] [--timeout TIMEOUT]

Named Arguments

-d, --debug  Possible choices: CRITICAL, ERROR, WARNING, INFO, DEBUG
          Default: “INFO”
-s, --source  Services topology source (use ‘help’ value to print available sources)
-f, --filter  Add endpoints filter (use ‘help’ value to print available filters)
-o, --output
Possible choices: services, zonefile, object-debug, endpoints, arc-CEs
Write requested data to stdout

--json
Change output format from plaintext to JSON
Default: False

--output-all
Output all services/endpoints including inactive (filters are still applied)
Default: False

-u, --ddns-update
Invoke DNS zone incremental DDNS update secured by TSIG key
Default: False

--domain
Domain name of the ARCHERY endpoint to use (required for DDNS update)

--ddns-master-ip
Master DNS IP address (required for DDNS update)

--ddns-tsig-keyfile
TSIG keyfile (required for DDNS update)

--ddns-tsig-algorithm
Possible choices: HMAC-SHA512, HMAC-SHA256, HMAC-MD5, HMAC-SHA1, HMAC-SHA384, HMAC-SHA224
Cryptographic algorithm for TSIG
Default: “HMAC-MD5”

--ttl
DNS resource records TTL value to use (default is 3600)
Default: 3600

--threads
Number of threads to fetch information in parallel (default is 8)
Default: 8

--timeout
Per-source information fetching timeout (default is 10 seconds)
Default: 10

3.3 Deploying Other ARC Products

3.3.1 ARCHERY Documentation

Deploying ARCHERY for Country/Project/VO

This document shows the example configuration of ARCHERY to hold Computing Elements endpoints for some project (country-level, experiment, virtual organization, etc).

Configuration examples in this guide are provided for BIND, however you can use any name server implementation.

Entry point

An entry point to ARCHERY-based registry is a domain name. In this guide example.org used as an entry point. To submit job using ARCHERY run:

```
arcsub -g example.org myjob.xrsl
```

The nordugrid.org represents the whole Nordugrid infrastructure entry point. It holds references to the country-level endpoints named upon country code top-level domain (ccTLD) identifiers: <ccTLD>.archery.nordugrid.org
Transaction signature key to manage endpoints

It is possible to manage records manually, however to provide up-to-date information and keep it simple to manage endpoints it is advised to use Dynamic DNS updates with archery-manage.

Generate key

Use the following command to generate the key:

```
dnssec-keygen -a HMAC-MD5 -b 256 -n USER archery
```

From the generated files you need a secret part to be included in both BIND and archery-manage configuration.

Define key in BIND

Create the /etc/named/archery.key and put the secret key generated on the previous step here:

```
key archery_key {
  algorithm hmac-md5;
  secret "S0Me+SecRet+keYgener@tedwithdnssec==";
};
```

Include key definition into named.conf using the config line:

```
include "/etc/named/archery.key";
```

Keyfile for archery-manage

Create a file (let it be named archery-manage.key) with the generated key in the following format:

```
archery_key:S0Me+SecRet+keYgener@tedwithdnssec==
```

Configure DNS zone to host endpoints

It is possible to put records directly into the example.org zone, however for manageability and security reasons it is advised to have dedicated zone configured for ARCHERY records.

You can use any name for dedicated zone, this example uses index.example.org.

Addresses used in the following examples:

- Primary (master) DNS: ns1.example.org (192.0.2.100)
- Secondary (slave) DNS: ns2.example.org (192.0.2.200)

Define zone in DNS

Add zone definition to master named.conf:

---

1 The content will be updated dynamically and it is better to eliminate the possibility of changing non-archery related records. Dedicated subzone is an easiest way.
2 Use <ccTLD>.archery.nordugrid.org if you are setting up country-level index for the Nordugrid infrastructure

---

3.3. Deploying Other ARC Products 169
Please note allow-update directive that authorize dynamic DNS update queries signed by archery-manage key.

Secondary DNS should be configured without any special options:

Create zonefile with basic structure

Zonefile requires only basic SOA record and will be filled with data by archery-manage. The zonefile template (timers are subject to arrange depending on planned update frequency):

```
$ORIGIN example.org.
$TTL 3600
index IN SOA ns1.example.org. hostmaster.example.org. (  
2018082401 ; serial  
1200 ; refresh (20 minutes)  
180 ; retry (3 minutes)  
604800 ; expire (1 week)  
60 ; minimum (1 minute)  
)  
NS ns1.example.org.  
NS ns2.example.org.  
```

Define records in parent zone

Note: If you setup country-level index for the Nordugrid infrastructure such records are defined in the parent nordugrid.org zone. Please provide your setup information for the top-level index instead of this section.

Define NS records\(^3\) to refer defined subzone:

\(^3\) If you plan to use different out-of-scope domain names in NS don’t forget to add glue A records.
To create an ARCHERY entry point in the parent zone you can:

- define CNAME record to use example.org as an entry point:

```
$ORIGIN example.org.
# ARCHERY entry point
_archery CNAME _archery.index
```

- OR define TXT resource record with ARCHERY data pointing to group:

```
$ORIGIN example.org.
# ARCHERY entry point
_archery TXT "u=index.example.org t=archery.group"
```

The same techniques can be used to define any other entry points (e.g. in the complete different domain). CNAME is recommended if you point to the only one ARCHERY group.

**Prepare static list of Computing Elements**

The initial list of CEs used by archery-manage to fetch the endpoints from LDAP GLUE2 and push it to DNS zone with incremental DDNS updates. CEs are specified line-by-line in the file:

```
ce01.example.org
ce02.example.org
arc6.example.org
```

To migrate from EGIIS archery-manage allows you to dump list of CEs automatically:

```
archery-manage -s egiis:ldap://old.egiis.example.org:2135/Mds-Vo-Name=ExampleVO, -o=grid \n    -o arc-CEs > ces.list
```

**Populate DNS with information**

To populate DNS zone with endpoints information you should run archery-manage in the following way:

```
archery-manage -s arcce-list:ces.list --ddns-update \n    --domain index.example.org --ddns-master-ip 192.0.2.100 \n    --ddns-tsig-keyfile archery-manage.key
```

**Note:** Updates are performed over the network, so you can run archery-manage on any host. It SHOULD NOT be DNS server itself.

 Consider to add filters to archery-manage:

**Filter: Port connectivity**

Check network connectivity to endpoint TCP port and filter endpoints that do not pass this test:

```
archery-manage -s arcce-list:ce.list -f portscan ...
```
Filter: Endpoint type

By default any endpoints available on CE are published. If you want to filter endpoints based on type there is a dedicated filter.

For example, if you want only EMI-ES ResourceInfo endpoints for EMI-ES only operation:

```
archery-manage -s arcce-list:ce.list -f type:org.ogf.glue.emies.resourceinfo ...
```

EMI-ES ResourceInfo and LDAP GLUE2 endpoints:

```
archery-manage -s arcce-list:ce.list -f type:org.ogf.glue.emies.resourceinfo,org.nordugrid.ldapglue2 ...
```

There is a special endpoint filter `resourceinfo` that is equivalent to EMI-ES ResourceInfo, LDAP GLUE2 and LDAP Nordugrid.

Filter: VO

For project-based ARCHERY deployment it is also useful to filter endpoints based on VO access policy. Only endpoints that advertise specified VO support will be added:

```
archery-manage -s arcce-list:ce.list -f vo:exampleVO ...
```

Checking the operation

General DNS queries

Any DNS client tool can be used, for example:

```
[user ~]$ host -t TXT _archery.example.org
_archery.example.org is an alias for _archery.index.example.org
_archery.index.example.org descriptive text "u=dns://ce01.example.org._archery._index.example.org. t=archery.service"
_archery.index.example.org descriptive text "u=dns://ce02.example.org._archery._index.example.org. t=archery.service"
<output omitted>

[root ~]$ host -t TXT ce01.example.org._archery.index.example.org
ce01.example.org._archery.index.example.org descriptive text "o=service t=org.nordugrid.arex id=ce01.example.org"
ce01.example.org._archery.index.example.org descriptive text "u=ldap://ce01.example.org. t=org.nordugrid.ldapglue2"
ce01.example.org._archery.index.example.org descriptive text "u=https://ce01.example.org. t=org.ogf.glue.emies.resourceinfo"
<output omitted>
```

Submit job

Use `arcsub -g example.org -d DEBUG` to check the process of endpoints discovery from ARCHERY.

---

Resource information (GLUE2PolicyRule in GLUE2 and nordugrid-cluster-acl in Nordugrid LDAP) will be used as a source of supported VOs.
Setup regular updates to ARCHERY

To keep information about endpoints up to date setup a CRON job or Systemd Timer to run `archery-manage` regularly. In combination with at least port filtering this allows to eliminate stale endpoints and actualize information in registry.

Configure LDAP-monitor to use ARCHERY

NorduGrid LDAP monitor support fetching endpoints\(^5\) from ARCHERY with the following configuration in `settings.inc`:

```php
$archery_list = array (  
    array (  
        "endpoint" => "example.org",
    )
);  
```

See also:

[ARCHERY Architecture Overview](#) ARCHERY Architecture Overview, including concept and DNS records specification.

[ARCHERY NorduGrid Top-Level Deployment](#) ARCHERY Deployment for Top-Level Nordugrid Registry, including example configuration.

---

\(^5\) Nordugrid monitor only works with `org.nordugrid.ldapng` data (and LDAP GLUE2 is experimental with known issues), so Nordugrid LDAP schema publishing is mandatory to use monitor.
CHAPTER FOUR

TECHNICAL DOCUMENTS DESCRIBING ARC COMPONENTS

Following documents gives a deep technical description of the various ARC components. If you are looking for architecture internals (how parts of ARC was designed) you can follows this section.

4.1 ARC Data Services Technical Description

4.1.1 ARC Data Services Overview

**Warning:** Whiteboard sketces to be replaces by series of digital figures

![Fig. 4.1: Using A-REX cache (stand-alone A-REX installation)](image)

4.1.2 A-REX Data Cache technical description

**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 srm://srm.nordugrid.org/grid/atlas/file1:
- is mapped to /cache/data/37/b19acc950c37876a61d2de6e238d38c9e94c0,
- the file /cache/data/37/b19acc950c37876a61d2de6e238d38c9e94c0.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.
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.

**Warning:** 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:** 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 modifiable by running jobs. This is potentially dangerous and so caching 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 caching.

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.

**Cache cleaning**

When [arex/cache/cleaner] block is defined 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, the option calculatesize=cachedir should be set in arc.conf so that the cache limits are applied on the size of the cache rather than the file system.

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.

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.

**Exposing the Cache**

Normally the ARC cache is internal to the CE and is not exposed to the outside. However it may be beneficial to allow reading cache files, if for example the file is lost from Grid storage or as a fallback when Grid storage is down. This can be done via HTTPS through the A-REX web services interface.

Specifying `arex/ws/cache` block opens remote read access to certain cache files for certain credential properties. When configured this allows cached files to be read from the A-REX WS endpoint, for example if file `gsiftp://my.host/file1` is cached at CE `a-rex.host` the file is accessible (if credentials allow) at:

```
https://a-rex.host/arex/cache/gsiftp://my.host/file1
```

Since remote reading can increase the load on A-REX, the number of concurrent requests should be limited. This can be done using the `max_data_transfer_requests` configuration option.

**Indexing the Cache content**

The **ARC Cache Index (ACIX)** provides a way to discover locations of cached files.

### 4.1.3 A-REX data transfer framework (DTR) technical description

This page describes the data staging framework for ARC, code-named DTR (Data Transfer Reloaded).

**Overview**

ARC’s Computing Element (A-REX) performs the task of data transfer for jobs before and after the jobs run. The requirements and the design steps for the data staging framework are described in *DTR Desing and Implementation Details*. The framework is called DTR (Data Transfer Reloaded) and uses a three-layer architecture, shown in the figure below:

The Generator uses user input of tasks to construct a Data Transfer Request (also 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. See *DTR priority and shares system* for more information.

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).

**Implementation**

The middle and lower layers of the architecture (Scheduler, Processor and Delivery) are implemented as a separate library `libarcdatastaging` (in `src/libs/data-staging` in the ARC source tree). This library is included in the `nordugrid-arc` common libraries package. It depends on some other common ARC libraries and the DMC modules (which enable various data access protocols and are included in `nordugrid-arc-plugins-*`).
packages) but is independent of other components such as A-REX or ARC clients. A simple Generator is included in this library for testing purposes. A Generator for A-REX is implemented in `src/services/a-rex/grid-manager/jobs/DTRGenerator.(h|cpp)`, which turns job descriptions into data transfer requests.

**Configuration**

Data staging is configured through the `[arex/data-staging]` block in `arc.conf`. Reasonable default values exist for all parameters but the `[arex/data-staging]` block can be used to tune the parameters, and also enable *multi-host data staging*. A selection of parameters are shown below:
### Parameter Explanation | Default Value
--- | ---
maxdelivery | Maximum delivery slots | 10
maxprocessor | Maximum processor slots per state | 10
maxemergency | Maximum emergency slots for delivery and processor | 1
maxprepared | Maximum prepared files (for example pinned files using SRM) | 200
sharetype | Transfer share scheme (dn, voms:vo, voms:group or voms:role) | None
definedshare | Defined share and priority | _default

### Multi-host related parameters

delivery-service | URL of remote host which can perform data delivery | None
localdelivery | Whether local delivery should also be done | no
remotesizelimit | File size limit (in bytes) below which local transfer is always used | 0
use-hostcert | Whether the host certificate should be used in communication with remote delivery services instead of the user’s proxy | no

Description of other data-staging parameters can be found in `[arex/data-staging]` block. The multi-host parameters are explained in more detail in *ARC Data Delivery Service Technical Description*

Example:

```
[arex/data-staging]
maxdelivery = 10
maxprocessor = 20
maxemergency = 2
maxprepared = 50
sharetype = voms:role
definedshare = myvo:production 80
deliveryservice = https://spare.host:60003/datadeliveryservice
localdelivery yes
remotesizelimit = 1000000
```

### Client-side priorities

To specify the priority of jobs on the client side, the `priority` element can be added to an XRSL job description, eg:

```
("priority" = "80")
```

For a full explanation of how priorities work see *DTR priority and shares system.*

### gm-jobs -s

The command “gm-jobs -s” to show transfer shares information now shows the same information at the per-file level rather than per-job. The number in “Preparing” are the number of DTRs in TRANSFERRING state, i.e. doing physical transfer. Other DTR states count towards the “Pending” files. For example:
As before, per-job logging information is in the controldir/job.id.errors files, but A-REX can also be configured to log all DTR messages to a central log file in addition through the logfile parameter.

**Using DTR in third-party applications**

*ARC SDK Documentation* gives examples on how to integrate DTR in third-party applications.

**Supported Protocols**

The following access and transfer protocols are supported. Note that third-party transfer is not supported.

- file
- HTTP(s/g)
- GridFTP
- SRM
- Xrootd
- LDAP
- Rucio
- S3
- ACIX
- RFIO/DCAP/LFC (through GFAL2 plugins)

**Multi-host Data Staging**

To increase overall bandwidth, multiple hosts can be used to perform the physical transfers. See *ARC Data Delivery Service Technical Description* for details.

**Monitoring**

In A-REX the state, priority and share of all DTRs is logged to the file controldir/dtr.state periodically (every second). This can then be used by the Gangliarc framework to show data staging information as ganglia metrics.

**Advantages**

DTR offers many advantages over the previous system, including:

- **High performance** When a transfer finishes in Delivery, there is always another prepared and ready, so the network is always fully used. A file stuck in a pre-processing step does not block others preparing or affect any physical transfers running or queued. Cached files are processed instantly rather than waiting behind those needing transferred. Bulk calls are implemented for some operations of indexing catalogs and SRM protocols.

- **Fast** All state is held in memory, which enables extremely fast queue processing. The system knows which files are writing to cache and so does not need to constantly poll the file system for lock files.

<table>
<thead>
<tr>
<th>Preparing/Pending files</th>
<th>Transfer share</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/86 atlas:null-download</td>
<td></td>
</tr>
<tr>
<td>3/32 atlas:production-download</td>
<td></td>
</tr>
</tbody>
</table>
Clean When a DTR is cancelled mid-transfer, the destination file is deleted and all resources such as SRM pins and cache locks are cleaned up before returning the DTR to the Generator. On A-REX shutdown all DTRs can be cleanly cancelled in this way.

Fault tolerance The state of the system is frequently dumped to a file, so in the event of crash or power cut, this file can be read to recover the state of ongoing transfers. Transfers stopped mid-way are automatically restarted after cleaning up the half-finished attempt.

Intelligence Error handling has vastly improved so that temporary errors caused by network glitches, timeouts, busy remote services etc are retried transparently.

Prioritisation Both the server admins and users have control over which data transfers have which priority.

Monitoring Admins can see at a glance the state of the system and using a standard framework like Ganglia means admins can monitor ARC in the same way as the rest of their system.

Scaleable An arbitrary number of extra hosts can be easily added to the system to scale up the bandwidth available. The system has been tested with up to tens of thousands of concurrent DTRs.

Configurable The system can run with no configuration changes, or many detailed options can be tweaked.

Generic flexible framework The framework is not specific to ARC’s Computing Element (A-REX) and can be used by any generic data transfer application.

Open Issues

- Provide a way for the infosys to obtain DTR status information
  - First basic implementation: when DTR changes state write current state to .input or .output file
- Decide whether or not to cancel all DTRs in a job when one fails
  - Current logic: if downloading, cancel all DTRs in job, if uploading don’t cancel any
  - Should be configurable by user - also EMI execution service interface allows specifying per-file what to do in case of error
- Priorities: more sophisticated algorithms for handling priorities
- Advanced features such as pausing and resuming transfers

Related Documents

DTR Design and Implementation Details

This page documents the requirements and design stages of the new data staging framework which took place around mid-2010.

Issues with previous implementation

1. Queueing happens per job - that makes it impossible to use potentially more effective processing order
2. Slow data transfers block those which could be done faster.
3. Big data transfers block small ones.
4. Jobs waiting for already cached (or to be cached) files are blocked by other jobs in queue.
5. Special features of sophisticated protocols are not taken into account - like SRM’s “try later”.
6. No priorities aka flexible queues.
7. No support for different credentials for different files.
8. No bandwidth handling.
9. No handling of files with different access latency (e.g., tape vs. disk)
10. No mechanism to choose a preferred replica for the LFC (catalog) inputs, for example: if replicas are in ndgf, swegrid, signet, unige, try with ndgf first.

Task Summary

The initial task was NOT to solve all these issues. The task was to create a framework which could be extended to solve them later, or to find/adopt such a framework.

Requirements

1. Effective usage of bandwidth. Whenever any transfer is paused due to any reason (tape stage in, retry later) for estimated time another transfer should use available bandwidth.
2. Transfer negotiation (with protocols such as SRM) should be independent of physical data transfer.
3. Each transfer should be capable to use own credentials.
4. Transfer should be capable of pausing (temporary cancel) and resuming (if protocol allows).
5. Automatic/dynamic redistribution of bandwidth is needed to allow short transfer to pass through even while big transfers are taking whole bandwidth.
6. Transfer from multiple alternative locations.
7. Cache checks should happen independently of data transfer to avoid locks.
8. Jobs where all files are cached should be processed immediately.
9. Better description of file source/destination than just URL (options are difficult to handle, something nicer is needed)
10. Priorities at different levels: among user groups, inside groups. Any other levels? 3 possible levels: among VO, users/roles inside VO, inside user identity.
11. Ability for users to set relative priority of their own jobs, both before submission and while job is in queue.

Security Requirements

• It must be built into the design that no process has higher privileges than necessary
• Elevated privileges are required for:
  – Access to cache. Cache files are only read/writeable by root user so they cannot be modified by jobs
  – Access to session and control directories. Access to the these directories should be performed under the uid of the job owner. The current method of running as root and chown’ing must not be used.
• Elevated privileges are not required for any other parts of the system such as scheduling

Performance Requirements

• Must scale higher than current highest workloads
• Must be able to handle up to 10000 active jobs (between ACCEPTED and FINISHED)
• Must be able to handle up to 1000 active physical transfers whilst ensuring all available bandwidth is used
• Must be able to handle transfers which have to wait for several hours before the physical file is ready to transfer

Possible solution to URL options problem

There has been a long standing problem with the format and syntax used to express URLs and associated metadata and options. While not directly related to data staging, it will be addressed as part of the data staging work.

Architecture Proposal

• 3 layers:
  1. Higher layer processing data staging requirements of jobs, collecting requested attributes/properties, resolving priorities (flattening them), managing credentials.
  2. Middle layer schedules individual transfers, suspends and resumes them, distributes bandwidth, etc.
  3. Lower level handles individual transfer protocol, communicates with middle layer to acquire, release and pre-allocate resources (mostly bandwidth), caches connections (if possible).

• Any layer can be outsourced to external service, for example gLite FTS

• Basic file staging stages:
  1. Identify transfer options
  2. Check cache
  3. Evaluate authorization (may require bandwidth)
  4. Resolve location (meta-URL, if needed, may require bandwidth)
  5. If needed repeat all steps from beginning
  6. Do transfer
  7. Post-processing (eg register replicas, release cache locks)

Requirements for components interfaces (based on protocol descriptions and architecture)

DTR Description

DTR stands for Data Transfer Request. This is the structure that contains several fields that fully describe the file transfer to be performed. One DTR is generated by the generator per each file transfer.

A detailed description and state diagrams of DTRs can be found in the Detailed description of DTRs

DTR Generator to DTR Scheduler

• Push DTR from Generator to Scheduler:
  – DTR contains description of single entity to be transferred - uploaded or downloaded. That includes endpoint, transfer parameters and limits, credentials - possibly multiple, etc.
  – Multiple DTRs may be affiliated together. Possible reasons and uses:
    * Belong to same job
    * Belong to bunch of jobs which user indicated as preferably processed together
    * Belong to same VO and assigned priorities to be applied within group
Fig. 4.4: Functional components and layers.
* Failure of one DTR in group may cancel processing of other DTRs (not sure, may be implemented in Generator)
  - DTR may have assigned priorities levels. Probably related to groups.
* Receive DTR from Scheduler to Generator:
  - Returned DTR indicates outcome of processing, either positive or negative. In last case it includes description of encountered problems and level of severity.
* Cancel DTR in Scheduler
* Modify DTR properties in scheduler. Possible usage: - Manipulate priorities

Generator

Job1 arrives

Push group of DTRs

Job2 arrives

Push group of DTRs

Job1 is canceled

Cancel DTRs in group 1

Job2 fails

Report finished DTRs

Schedule

DTRs are put into queue

DTR1 DTR2 DTR3

DTRs are put into queue

DTR1 DTR4 DTR5 DTR2 DTR3

DTRs are removed

DTR1 DTR4 DTR5

DTR1 and 4 are finished

DTR5 failed

Fig. 4.5: Example of communication session between DTR Generator and DTR Scheduler.

DTR Scheduler to DTR Preprocessor

* Push DTR from Scheduler to Preprocessor
  - Because DTR preprocessing is supposed to take short time it may include processing timeout
* Receive DTR from Preprocessor to Scheduler
  - Returned DTR indicates outcome of processing
    * Positive
      - DTR comes with information need for further processing either in Preprocessor or Delivery unit
      - DTR may contain multiple/alternative additional endpoints
      - Probably such DTR may be presented in tree-like diagram
    * Failure - includes description of encountered problems and level of severity
    * Delayed processing
- Includes retry time and possible margins.
- Scheduler must ensure this DTR will go back to Preprocessor within specified time margins.

- Cancel DTR in Preprocessor

Fig. 4.6: Example of communication session between DTR Scheduler and DTR preprocessor.

**DTR scheduler to DTR delivery**

- Push DTR from Scheduler to Delivery
  - DTR may have bandwidth assigned
  - DTR may have timeout related parameters assigned - minimal transfer rate, maximal inactivity timeout, etc.
- Cancel DTR in Delivery
- Suspend DTR in Delivery - should DTR leave Delivery or should it stay there?
- Receive DTR from Delivery to Scheduler. Returned DTR indicates outcome of processing
  - Positive
- Partially positive (partial data delivered)
- Redirection
- Failure - includes description of encountered problems and level of severity

- Get information about bandwidth currently used by Delivery
- Modify assigned bandwidth
  - May be used to free some bandwidth for urgent transfers

**Component Workflows**

**Generator**

The Generator is an integral part of the a-rex process. Internally it performs 4 tasks in following order:

- Makes DTRs out of existing job descriptions also assigning priorities and grouping information and makes them available to the Scheduler.
- Communicates immediate requests like DTR cancel or suspend to the Scheduler.
- Monitors DTR states as reported by the Scheduler (and possibly by other modules) in order to provide feedback to client tools of A-REX asking for job state.
- Receives finished/failed DTRs from the Scheduler and initiates job processing continuation. Note: This part may be merged with previous one.

**Scheduler**

“Queues” are queues of DTRs waiting to enter pre/post-processing or delivery. They are kept internal to the scheduler. The scheduler is the only place with complete knowledge of the system and the only place where priorities are controlled and error conditions are handled. When an event is received and involves sending the DTR to a queue, the DTR is put at a certain position in the queue. Depending on the DTR priority, other DTRs in the queue may be moved, paused or stopped to allow higher priority DTRs to proceed quicker or consume more resources. On receiving an event, the relevant queue is examined and action taken (eg if a delivery finished start a new one).

**Reactions to new events**

For simplicity error conditions are not included in the workflow here but described separately on the *Detailed description of DTRs*. They are examined by the scheduler, which will decide the workflow - do any necessary post-processing, decide whether to retry (immediately or after some delay) or to report back to the generator that the DTR definitively failed. DTR state transitions are also described in more detail and with diagrams on the *Detailed description of DTRs*.

- New DTR from generator
  - if cacheable or meta-protocol:
    * add to pre-processor queue (for cache check, replica resolution etc)
  - else if ready to be delivered (base protocol for src and dest):
    * add to delivery queue
- DTR returned from pre-processor
  - if cached:
    * send to post-processor to be linked
  - else if need more pre-processing:
Scheduler

Transfer request arrives

Push DTR

State, bytes transferred, used bandwidth, etc.

Urgent transfer request

Decrease bandwidth

Push DTR

State, bytes transferred, used bandwidth, etc.

Job is canceled

Cancel DTR1

State, bytes transferred, used bandwidth, etc.

Report DTR2 finished

Delivery

Periodic report about state of DTR1

Periodic report about state of DTR1 and DTR2

Periodic report about state of DTR2

No DTRs to report

Fig. 4.7: Example of communication session between DTR Scheduler and DTR Delivery.

4.1. ARC Data Services Technical Description 189
add to pre-processing queue

- else:
  * add to delivery queue

• DTR returned from delivery
  - if post-processing required (index registration, cache linking, release request):
    * put in post-processor queue
  - else:
    * return to generator

• DTR returned from post-processor:
  - Return to generator

• DTR cancel notification from generator
  - if before pre-processing:
    * return to generator
  - else if in post-processing:
    * wait until finished and then send back to clean up
  - else:
    * cancel immediately and add to post-processing queue for clean up

• DTR modify notification from generator
  - change request immediately, and modify queue if appropriate

Processor

The processor is divided into two logical parts: the pre-processor and the post-processor. Either part is invoked by the scheduler as a process/thread and has the DTR to process. Therefore, the pre- or post-processor can be a straightforward function, performing the next steps:

The pre-processor:

• check endpoint for its presence in cache
  - if successful, mark DTR as BYPASS_TRANSFER, return the DTR to the scheduler
  - if file is not in cache, construct cache endpoint for DTR destination and return to scheduler

• resolve the replicas of the file, if needed
  - return the DTR with a failure if no locations have been found
  - return the list of replicas found to scheduler
  - Note: The pre-processor doesn’t care if the resolved locations represent meta-protocols themselves, it’s the scheduler’s job to determine it and possibly send this DTR for the pre-processing once again.

• query an endpoint
  - supply information on size, checksum, creation date, access latency as available according to protocol

• If an asynchronous request needs to be performed, for example SRM prepareToGet
  - Start request, mark DTR as STAGING_PREPARING_WAIT, return to the scheduler. An estimated wait time may be set by the pre-processor from information supplied from the remote service.

• If a polling request arrives from the scheduler
  - Check state of asynchronous request and report back to scheduler success, wait more, or error.
• If the cancellation request arrives from the scheduler
  – interrupt the operation, mark the DTR as PREPROCESSING_CANCELLED, return to the scheduler

• If the preprocessor hits the timeout during performing these tasks
  – interrupt the operation, mark the DTR as PREPROCESSING_NOT_FINISHED, return to the scheduler.

Possible features of the pre-processor in the future

• for each of resolved locations:
  – check options one by one (we still have to define transfer options, below is the example), mark in the
    list to which extent it satisfies the option
    * can the location provide a required bandwidth
    * other options, specified for the user
    * keep processing all the options even if some of them are not satisfied – the scheduler may later
      review the options and start the processing again
  – request the file size (if possible) and compute estimated transfer time, mark it in the list for this location

• if there are in the list locations that satisfied all the checks
  – return the DTR as TRANSFER_READY to the scheduler with the list of these locations and their
    estimated transfer times

• if there are no locations in the list that satisfied all the checks
  – return the DTR with the least severe encountered failure (from most severe to least severe, Location
    is dead - Location is not authorized - Location doesn’t satisfy the options), so the scheduler can either
    drop the DTR (in case of dead/non-authorized storages) or review transfer options and try again (in
    case of unsatisfied options)

The Post-processor:

• release stage requests;
• register a replica in an index service;
• release cache locks;

Delivery

Delivery is a component where the transfer orders are handled. Delivery process is listening to the Scheduler

to receive the ready DTRs for transferring or any other events such as decreasing the bandwidth or suspending
and canceling the ongoing jobs. Delivery process, reports the status of the DTRs and events to the scheduler
periodically.

• Transfer request delivery
  – Pick up a DTR from delivery queue.
  – Check the source and destination, set the bandwidth, timeout and etc.
  – Start transferring by placing the received data in a buffer.
  – Periodically report the status of DTR such as transferred bytes, used bandwidth to Scheduler.
  – By finishing the data transfer do a checksum
    * if checksum is correct return a SUCCESS status to the Scheduler
    * else retry transfer again

• High priority transfer request event
  – Change the DTR to use max bandwidth n
• Job suspension
  – Suspend a transfer request
  – Keep the transferred data information in the delivery process
  – Release the used bandwidth
  – Report the status of suspended DTR to Scheduler

• Transferring cancel
  – Stop transferring
  – Release the resources if any is in use
  – Clean up the buffer

• Status report to Scheduler
  – Periodically collect the information of the DTRs in progress.
  – Calculate used bandwidth, transferred data bytes, status
  – Report the status to the Scheduler

Protocol Interfaces

Current Interface

The following interface is defined for each protocol through the DataPoint class:

• StartReading - Start reading data from the source
• StopReading - Stop reading data from the source
• StartWriting - Start writing data to destination
• StopWriting - Stop writing data to destination
• Resolve - Find physical replicas from an indexing service
• Check - Verify that the current credentials have access
• ListFiles - Find metadata
• Remove - Delete
• (Pre)Register and (Pre)Unregister - add and delete in indexing service

These do not fulfil all the requirements of the DTR interfaces described above.

New Interface

The main limitation of the current interface is that it does not handle efficiently those protocols such as SRM which involve asynchronous preparation steps. In the new framework, having to wait for a file to be prepared should not block other activities. Therefore the proposal is to split protocols as now into meta and direct protocols, but introduce a third class of stageable protocols and appropriate extra methods to handle them. A stageable protocol could also be a meta protocol or a direct protocol. Extra methods are also needed to handle pausing and cancellation of transfers.

For Meta Protocols (eg LFC, RLS)

• Resolve - Resolve replicas in an indexing service
• (Pre)Register and (Pre)Unregister - add and delete in indexing service
For Stageable Protocols (eg SRM, Chelonia)

- **PrepareReading** - Prepare the storage service for reading. This may involve preparation of Transport URLs (TURLs) which should then be used for physical data transfer, or operations like reading data from tape to disk. If the protocol’s implementation is asynchronous then this method may return a status that tells the caller to wait and poll later. The caller should call this method again after some period of time (the remote service may provide an estimate of the preparation time).
- **PrepareWriting** - Prepare the storage service for writing. Works similarly to PrepareReading.
- **FinishReading** - Release or abort requests made during PrepareReading, usually called after physical transfer has completed.
- **FinishWriting** - Release or abort requests made during PrepareWriting, usually called after physical transfer has completed.
- **Suspend** - Pause current preparation of transfer
- **Resume** - Resume suspended preparation of transfer (Note: depending on the protocol, suspend and resume may be implemented as stop current request and start new request)
- **Cancel** - Stop preparation of transfer

For Direct Protocols (eg FTP, HTTP)

- **StartReading** - Start reading data from the source
- **StopReading** - Stop reading data from the source
- **StartWriting** - Start writing data to destination
- **StopWriting** - Stop writing data to destination
- **Modify** - Change parameters of transfer such as bandwidth limits
- **Suspend** - Pause current transfer
- **Resume** - Resume suspended transfer
- **Cancel** - Stop current transfer

For All Protocols

- **Check** - Verify that the current credentials have access
- **List** - Find metadata
- **Remove** - Delete

Existing ARC Code

We should aim to re-use as much as possible of the existing ARC code. The arc1 code base will be used for all developments. Code for job description handling and interfacing with the LRMS can remain unchanged, all that concerns us is the code handling the PREPARING and FINISHING states. Job state handling is done in states.cpp. This code can remain largely unchanged but in the ActJobPreparing/Finishing methods the job enters the new system at the upper layer.

The lower level code for data transfer for each protocol is handled in the DMCs. These can largely remain unchanged except for the extra methods in the meta-protocols above.

Caching code can remain unchanged.
PREPARING/FINISHING State Semantics

The semantics of these states may need to be changed - at the moment PREPARING/FINISHING means that the job is transferring data and PENDING those states means a job cannot enter the state due to some limit. In our new system there is less of a distinction between jobs waiting and jobs transferring data, also some files within the job may be transferring while some are waiting. Once the job enters the upper layer of the new system it will be in a staging state even though it may have to wait a long time before any data is transferred.

Processes and Threads

In the current architecture a persistent A-REX thread is spawned by HED. A new data staging process (downloader/uploader) is forked for each job to stage its input or output files. Within each process a thread is created per file. The status of each thread is reported back to the loader process through callbacks and the process exit code tells the A-REX the status of all transfers. There are some problems with this approach:

- The A-REX knows nothing about what happens inside the loader until the process exits and cannot communicate with the loader
- It is not possible to change the uid of each thread individually so they all run under the uid of the downloader/uploader. The uid of the downloader/uploader depends on configuration. If possible uid of mapped user is used. But if there is cache shared among users that uid will be root or uid of A-REX.

To solve this last problem, processes in the delivery layer writing to or from the session dir must run under the locally mapped uid. Writing to the cache must always be done as root and so cache and non-cache downloads must be done in different processes. This leads to the conclusion that the delivery layer must be separate processes from the scheduling layer. Then to solve the first problem there needs to be a method of communication between the delivery and scheduling layers, which must be two way, so that the scheduler can modify on-going transfers and the transfer can report its status back to the scheduler.

Notes on running processes under mapping uid:

- It is only open() operation which needs to be run under special uid. If we follow convention that all open() operations in an executable are called through single wrapper function and put global lock around it, then we can have filesystem access under selected uid inside multi-threaded application. Unfortunately for NFS the process needs to maintain the uid throughout the whole transfer.
- This may introduce performance issue if open() operation takes too long, like in case of remote file system.
- open() may be called by an external library which we have no control over
- according to ‘man open’ “UID mapping is performed by the server upon read and write requests” and hence suggested approach will fail on NFS.

Operations carried out by the pre- and post-processor require access to the filesystem in the following steps

- Cache preparation - locking the cache file and checking its existence. This must be done as root.
- Cache finalisation - copying or linking the cached file to the session directory. The hard link to the per-job dir must be done as root but the soft linking or copying to the session dir must be done as the mapped user.
  - Last is rather “must” than “may” to make it work on NFS with root_squash=on. The situation may exist where cache is on NFS and then process copying file must switch uid while accessing file and its copy.
- Access to proxy - the proxy is needed when contacting secure remote services. The proxy in the control dir is owned by the mapped user. Therefore either we have to:
  - Make a copy of the proxy owned by root - this does not fit our security requirements above
    * Note1: This is how it is done in current implementation. There is no security hole here because this is same proxy which was obtained by A-REX and written to control directory under root uid. So this proxy already belonged to root and making belong to root again makes little difference.
    * Note2: Because as a rule proxy is stored on local file system it is always accessible by root. Copying of proxy was needed in current implementation due to limitation of Globus libraries -
those were able to accept proxy only from file and were checking if proxy belonged to current
uid. Because ARC data library allows assigning credentials to communication directly (at least it
should, but may be not implemented for some protocols, needs checking) and because proxy is
readable by root such trick is not needed anymore.

– Use processes rather than threads for the pre- and post-processor, changing the uid of the process to
the mapped user

  * This approach is most probability not needed for this purpose but may be very much desirable for
  fighting NFS.

– As suggested above, use global open() function to open with mapped id then switch back to root - this
suffers the same problems mentioned above

  * There is no need to do that for proxy because root can always open files belonging to other users

The generator will be a thin layer between the main A-REX process which moves jobs from state to state and the
scheduler which processes DTRs - when a job enters the PREPARING or FINISHING state the generator will
create DTRs and send them to the scheduler. For the scheduler to be efficient it should run as a separate process
(or thread). Each pre-processing step such as resolving replicas in a catalog or a cache permission check should
be fast (< few seconds) but should be run asynchronously in order not to block the scheduler if something bad
happens.

In summary we have the following processes corresponding to the three layers:

- Persistent main A-REX process (thread of HED) and persistent generator thread
- Persistent scheduler process (or thread) with temporary threads for pre-processing
- Temporary delivery processes created per file transfer

Using processes vs threads should add no more CPU load, however more memory is required. For increasing
performance it should be possible to reuse processes in a way similar to how threads are reused.

Implementation Choices

The current system uses files in the control directory for all communication. We may need something more
sophisticated for this more complex system, either internal to the new system or also to replace the current files in
control dir method. Possibilities:

- Files
- Sockets
- Database
- RPC
- Pipes
- Message passing
  - ActiveMQ - http://activemq.apache.org/
- ...

Suggestion 1

- Use persistent object - file, database record - for storing DTRs. Each object includes:
  - Description of DTR
  - Owner of DTR
• Last requested action
• Current state inside owner (or maybe inside every module).
• Each DTR has own ID - may be just file name or record key
• Keep simple communication channels between modules - like 2 pipes for example
  – Whenever DTR is changed its ID is sent over communication channel to module which is supposed to react
  – As backup modules can scan modification timestamps of objects periodically

Pros:
• Simple
• Persistency

Cons:
• Monitoring of ever changing state - like bandwidth usage - would require constant modification of files/records.
  – This problem could be solved by providing information which needs no persistency through communication channel. But that would make communication more complex.
  – Another possible solution is to mmap (should work well with files) objects and flush them to persistent store only if persistent part of information is modified.

Suggestion 2 (used in current implementation)

• DTR objects can be rather complicated, so keep them only in memory
• In case of process failure all DTRs are reconstructed from control files into initial NEW state
• DTRs are passed as objects between threads
  – Separate threads run for A-REX (including the Generator), Scheduler, and Delivery components
  – Delivery thread starts new processes for each transfer
    * These communicate simple status messages ("heartbeats") through pipes to main delivery thread
• Communication between threads through callbacks

Pros:
• Simplifies development - no need for complex persistency layer or serialisation code
• Fast communication through thread callbacks

Cons:
• Having lots of threads increases risks of deadlocks and race conditions
• No persistency
• No way to communicate from scheduler to transfer processes
  – If the only communication required is to pause/resume/cancel a transfer it can be done through signals eg SIGSTOP/SIGCONT/SIGTERM

Implementation idea for Suggestion 2

• Generator, Scheduler, Delivery processes are singletons and run as persistent threads. The Processor is a singleton.
Note: C++-wise it would be probably more correct to use static methods instead of singletons. With such approach where would be no need to handle singletons outside of class itself hence simplifying interface. Although inside class there may be singleton.

• A-REX initiates a request to the data staging system by calling Generator::receiveJob()
  – Generator creates DTRs and after they finish it modifies the job state directly
  – A-REX can query job status from Scheduler via Generator

• Generator communicates with scheduler by calling Scheduler::receive_dtr method
  – There is also a Scheduler::cancel_dtrs method which is called by the Generator to cancel DTRs

• Scheduler has an instance of DTRList class, which is the storage for all the DTRs in the system.
• Scheduler communicates (passes a DTR) to any other process by calling DTR::push() method
• Pre-, post-processor and delivery communicate with scheduler by simply changing the status of DTR and changing the owner to “SCHEDULER”. Scheduler will pick up DTRs with changed statuses automatically during next iteration.
• When DTR goes from the scheduler to the pre or post-processor, it calls the processDTR() method of the singleton processor within push(). The processor then spawns a thread and returns.
• When DTR goes from the scheduler to delivery, within push() it calls the processDTR() method of the singleton delivery. The delivery then spawns a process and returns.
  – The delivery singleton receives messages through pipes from the transfer processes and reports information in the DTR object.

Detailed description of DTRs

DTR stands for Data Transfer Request. This is the structure that contains several fields that fully describe the file transfer to be performed. One DTR is generated by the generator per each file transfer.

Fields of the DTR

More or less required:

• DTR ID
• source endpoint
• destination endpoint
  – for source and destination, a list of metadata such as file size, checksum, creation date etc
  – for source and destination (if applicable) a list of replicas
  – for source and destination (if applicable) current replica
  – for source and destination (if applicable) TURL or delivery-level URL used for transfer
  – for source and destination (if applicable) request ID (in the case of asynchronous requests to remote storage services)

• credentials
• cache information
  – if the file is cacheable, the filename in cache
  – cache directories configuration
  – caching state (already in cache, cache currently locked etc)

• local user information (uid/gid)
• Job ID this transfer belongs to
• priority of the transfer - a number set by the generator which flattens priorities
• transfer share this DTR belongs to
• sub-share the DTR belongs to - may be set by the Generator
• tries left
• flags to handle properties and strategies when dealing with index servers
  – flag to say whether DTR is replicating inside the same logical filename
  – flag to say whether DTR should force registration to an existing logical filename, if the source is different
• mapping info - mapping information of local files to which remote files may be mapped to in the configuration (copyurl/linkurl)
• status of the DTR
• error status
  – type of error
  – location of error
  – text description of error detail
• number of bytes transferred/offset
• timing properties
  – timeout - time which DTR is allowed to remain in current state
  – creation time
  – last modification time
  – process time - wait until this time to do further processing
• cancel (set to true if request is to be cancelled)
• bulk operation flags to combine several DTRs in a bulk request
• delivery endpoint, whether Delivery is to be carried out by a local process or remote service
• current owner - who is in charge for this DTR right now
• logger object, so each DTR can have its own log
• lock, since DTRs can be modified by several processes, for avoiding writing collisions

Possible
• affiliation (if we use the affiliation of multiple DTRs, see right below).
• history of states

Multiple DTRs may be affiliated together. Possible reasons and uses:
• Belong to same job
• Belong to bunch of jobs which user indicated as preferably processed together
• Belong to same VO and assigned priorities to be applied within group
• Failure of one DTR in group may cancel processing of other DTRs (should be implemented in Generator)

State transitions of DTR

All possible states of a DTR, with arrows indicating the normal flow of DTRs between states. Each state is explained in detail below. Error conditions are not included here but are shown in another diagram further down.
Fig. 4.8: DTR State Diagram
Status codes

The following table describes all non-error status codes, and also the action taken in the event of a cancellation request being received while in that state. In general if all of the data transfer has been completed before receiving a cancellation request, the destination file is not deleted. The main reason for this is to preserve cache files, as the user may wish to run the same job soon after cancelling it.

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Text Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Statuses set by the generator</strong></td>
<td></td>
</tr>
<tr>
<td>NEW</td>
<td>The DTR has just been built by the generator</td>
</tr>
<tr>
<td>CANCEL</td>
<td>A request has been made to cancel the DTR</td>
</tr>
<tr>
<td><strong>Statuses set by the scheduler</strong></td>
<td></td>
</tr>
<tr>
<td>CHECK_CACHE</td>
<td>The DTR destination is cacheable and the cache should be checked for the file's existence</td>
</tr>
<tr>
<td>RESOLVE</td>
<td>The DTR source is a meta-protocol and should be resolved</td>
</tr>
<tr>
<td>QUERY_REPLICA</td>
<td>The DTR source should be queried to check existence, check file size, checksum etc.</td>
</tr>
<tr>
<td>PRE_CLEAN</td>
<td>The destination in the DTR should be deleted before writing</td>
</tr>
<tr>
<td>STAGE_PREPARE_SOURCE</td>
<td>The DTR source is a meta-protocol which must be prepared or staged</td>
</tr>
<tr>
<td>STAGE_PREPARE_DESTINATION</td>
<td>The DTR destination is a meta-protocol which must be prepared or staged</td>
</tr>
<tr>
<td>TRANSFER_WAIT</td>
<td>The DTR is ready to be sent to delivery but must wait due to transfer limits or priority settings</td>
</tr>
<tr>
<td>TRANSFER</td>
<td>The DTR should be transferred immediately</td>
</tr>
<tr>
<td>RELEASE_REQUEST</td>
<td>The DTR transfer has finished and any requests made on remote storage should be released</td>
</tr>
<tr>
<td>REGISTER_REPLICA</td>
<td>The DTR destination is a meta-protocol and the new replica should be registered</td>
</tr>
<tr>
<td>PROCESS_CACHE</td>
<td>The DTR destination is cacheable and the cached file should be unlocked and linked/copied</td>
</tr>
<tr>
<td>DONE</td>
<td>The DTR completed successfully</td>
</tr>
<tr>
<td>CANCELLED</td>
<td>The DTR has been cancelled successfully</td>
</tr>
<tr>
<td>ERROR</td>
<td>An error occurred with the DTR</td>
</tr>
<tr>
<td><strong>Statuses set by the pre-processor</strong></td>
<td></td>
</tr>
<tr>
<td>CHECKING_CACHE</td>
<td>The pre-processor is checking the cache</td>
</tr>
<tr>
<td>CACHE_WAIT</td>
<td>The cache file is locked and the scheduler should wait before trying to obtain the lock</td>
</tr>
<tr>
<td>CACHE_CHECKED</td>
<td>The cache check is complete</td>
</tr>
<tr>
<td>RESOLVING</td>
<td>The pre-processor is resolving replicas</td>
</tr>
<tr>
<td>RESOLVED</td>
<td>The replica resolution is complete</td>
</tr>
<tr>
<td>QUERYING_REPLICA</td>
<td>The pre-processor is querying a replica</td>
</tr>
<tr>
<td>REPLICA_QUERIED</td>
<td>The replica querying is complete</td>
</tr>
<tr>
<td>PRE_CLEANING</td>
<td>The pre-processor is deleting the destination file</td>
</tr>
<tr>
<td>PRE_CLEARED</td>
<td>The destination file has been deleted</td>
</tr>
<tr>
<td>STAGING_PREPARING</td>
<td>The pre-processor is making a staging or preparing request</td>
</tr>
<tr>
<td>STAGING_PREPARING_WAIT</td>
<td>The staging or preparing request is not ready and the scheduler should wait before polling</td>
</tr>
<tr>
<td>STAGED_PREPARED</td>
<td>The staging or preparing request is complete</td>
</tr>
<tr>
<td><strong>Statuses set by the delivery</strong></td>
<td></td>
</tr>
<tr>
<td>TRANSFERRING</td>
<td>The transfer of the DTR is on-going</td>
</tr>
<tr>
<td>TRANSFERRED</td>
<td>The transfer completed successfully</td>
</tr>
<tr>
<td><strong>Statuses set by the post-processor</strong></td>
<td></td>
</tr>
<tr>
<td>RELEASING_REQUEST</td>
<td>The post-processor is releasing a stage or prepare request</td>
</tr>
<tr>
<td>REQUEST_RELEASED</td>
<td>The release of stage or prepare request is complete</td>
</tr>
<tr>
<td>REGISTERING_REPLICA</td>
<td>The post-processor is registering a replica in an index service</td>
</tr>
<tr>
<td>REPLICA_REGISTERED</td>
<td>Replica registration is complete</td>
</tr>
<tr>
<td>PROCESSING_CACHE</td>
<td>The post-processor is releasing locks and copying/linking the cached file to the session dir</td>
</tr>
<tr>
<td>CACHE PROCESSED</td>
<td>Cache processing is complete</td>
</tr>
</tbody>
</table>
Error Conditions of DTRs

The following diagram shows possible error conditions and actions taken. For simplicity and because all error handling logic takes place within the scheduler, the pre- and post-processor and the delivery layers are not shown. Errors are categorised into the following types:

<table>
<thead>
<tr>
<th>Error</th>
<th>Explanation</th>
<th>Retryable?</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTER-INTERNAL_LOGIC_ERROR</td>
<td>Internal error in data staging logic</td>
<td>No</td>
<td>Stop processing and report back to generator</td>
</tr>
<tr>
<td>INTER-INTERNAL_PROGRESS_ERROR</td>
<td>Internal error like losing contact with an external process</td>
<td>Yes</td>
<td>Clean if necessary and retry</td>
</tr>
<tr>
<td>SELF-REPLICATION_ERROR</td>
<td>Attempt to copy a file to itself</td>
<td>No</td>
<td>Return to generator</td>
</tr>
<tr>
<td>CACHE_ERROR</td>
<td>Problem occurred in cache handling</td>
<td>Yes</td>
<td>Retry without caching</td>
</tr>
<tr>
<td>TEMPORARY_REMOTE_ERROR</td>
<td>Error such as connection timeout on remote service</td>
<td>Yes</td>
<td>Retry with an increasing back-off</td>
</tr>
<tr>
<td>PERMANENT_REMOTE_ERROR</td>
<td>Error such as file not existing, permission denied etc on remote service</td>
<td>No</td>
<td>Follow cancellation steps and return failed DTR to generator</td>
</tr>
<tr>
<td>LOCAL_FILE_ERROR</td>
<td>Error with a local file</td>
<td>No</td>
<td>Follow cancellation steps and return to generator</td>
</tr>
<tr>
<td>TRANSFER_SPEED_ERROR</td>
<td>Transfer rate was below specified limits</td>
<td>Yes</td>
<td>Retry transfer. If all retries fail, report back to generator - it will make the decision on whether to cancel other related DTRs. (Future work: make decision on whether other transfers caused slow transfer and whether cancelling others would help or should be done)</td>
</tr>
<tr>
<td>STAGING_TIMEOUT_ERROR</td>
<td>The staging process took too long</td>
<td>No</td>
<td>Try a different replica - if none available, cancel and report back to generator</td>
</tr>
</tbody>
</table>

Methods of DTRs

DTR::push (DTR, receiver) – pass the DTR from one process to another, e.g. DTR::push (dtr, preprocessor)

Implementation

Within Data Staging framework there is a global list of DTRs. Pointers to the DTRs are passed around between components, which can modify them directly and push them between each other.

DTR priority and shares system

Here we describe the priority and shares system for the new data staging framework. During the design stage there were several ideas taken from other research in the field and the first implementation of the transfer shares model in ARC.
Fig. 4.9: DTR Error State Diagram
Ideas behind priorities and fair-share in data staging

The initial idea was giving every DTR that comes into the system a fixed priority, then sorting the queue according to priorities and launching the first N DTRs with the highest priorities. This scheme also allows easy incorporation of pre-emption: if the job with higher priority appears it just pushes other DTRs out of the front of the queue and then during the next scheduler loop we can start these DTRs and suspend the pushed ones.

However, this idea can potentially lead to the situation that demanded the implementation of transfer shares in ARC. If a user or VO with the highest priority submits bunch of jobs at once all the other will be blocked, because DTRs from this bunch will occupy the front of the queue for a long time.

The idea of transfer shares comes in handy now. The available transfer slots should be shared among different VOs/ Users, so nobody would be blocked. VOs/ Users with higher priority get more transfer slots than the other. However, strict limits on the number of slots per share are not flexible enough - if the transfer pattern changes then strict limits could cause problems, squeezing lots of users/jobs into one share with a few slots and blocking others. The User or VO must also be able to decide the relative priority of jobs within its own share.

Current Implementation

The ideas above led to the creation of two configurable properties: user-defined job priority and server-defined share priority. Users may define a priority for their jobs in the job description (“priority” attribute in xrs I), and this is a measure of the priority given to this job within the share it gets assigned to when submitted. On the server-side, it is possible to define a share type and priorities of certain shares. The share priority is used to determine the number of transfer slots to assign to the share, taking into account which shares are currently active (active meaning the share has at least one DTR in the system).

When the Scheduler receives a new DTR, it is placed into a transfer share, which is defined by a User DN, VO, Group inside VO or role inside VO as it was in previous versions of ARC. Currently it’s possible to use only one sharing criteria in the configuration, i.e. it’s not possible to use simultaneously sharing by User and VO.

Priority is defined as a number between 1 and 100 inclusive - a higher number is a higher priority. In the A-REX configuration it is possible to specify a base priority for certain shares. If the DTR doesn’t belong to any of these specified shares, it is placed in a “_default” share with default base priority (50). The scheduler sets the priority of the DTR to the base priority of the share multiplied by the user priority from the job description (default 50) divided by 100, therefore default priority of a DTR is 25. In this system the priority set in the job description effectively defines a percentage of the base priority. Thus service administrators can set maximum priority limits for certain shares, but users or VOs have full control of their jobs’ priority within the share.

While revising the Delivery and Processor queues, the scheduler separates DTRs according to the shares they belong to. Inside every share DTRs are sorted according to their priorities. Then the scheduler determines the number of transfer slots that every active share can grab. The number is determined dynamically depending on priorities of active shares. Each share receives the number of slots which corresponds to the weight of its priority in the summed priority of all active shares. After the number of slots for each share is determined the scheduler just launches N[i] highest priority DTRs in each share, where N[i] is the number of transfer slots for i-th share.

The reason for weighting the DTR priority by the share priority is for occasions when the Scheduler considers the entire queue of DTRs, for example when allowing highest priority DTRs to pass certain limits.

Example: there are two active shares, one has base priority 60, the other 40. The summarized priority is 100 (60 + 40). The first share has a weight of 60%, the second 40%. So the first will grab 60% of configured transfer slots, and the second – 40%. If the system is configured with 5 Delivery slots, then the first share will take 3 slots and the second 2 slots. The 3 highest priority DTRs from the first share and 2 highest priority from the second share will be assigned to those slots.

Emergency Shares

To avoid the situation where a fixed limit of slots are used up by slow transfers and a new high priority transfer has to wait for a slot, we have “emergency” transfer slots. If there are transfers in the queue from a particular share, but all slots are filled with transfers from other shares, one emergency slot can be assigned to this share to allow
transfers to start immediately. The share may use an emergency slot until any other transfer finishes, at which point the emergency slot becomes a regular slot and a new transfer does not start from the queue.

**Sub-shares**

The Generator can assign DTRs to “sub-shares” to give a higher granularity than the standard criteria and when assigning transfer slots. Sub-shares are treated as separate shares. In A-REX, different sub-shares are assigned to downloads and uploads, and in this case emergency transfer slots prove useful for preventing jobs not being able to finish because all transfer slots are taken by downloaders. If this happens then emergency slots can be used for uploads.

**Potential Problems**

- Within a share, high priority jobs block low priority jobs. Thus if there is a constant stream of high priority jobs in a share, then some low priority jobs in the same share may never run. Possible solutions:
  - Increasing the priority as the time spent in the queue increases (returning to previous priority after leaving the queue). This is currently implemented as increasing the priority by 1 every 5 minutes after the DTR’s timeout has passed.
  - Changing simple highest-priority-first to a random algorithm where higher priorities are weighted higher
  - Making a higher granularity of shares by splitting each priority or priority range into its own share - this is probably too complicated

**A-REX Configuration**

The configuration varies depending on the ARC version. In the examples below 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.

```plaintext
[arex/data-staging]
sharetype="voms:role"
definedshare="atlas:slow-prod 20"
definedshare="atlas:validation 80"
```

If both shares are active and there are 10 slots available, then DTRs in the slow-prod share will get 2 slots and those in the validation share get 8 slots, and so the jobs in the validation share will have a higher throughput (assuming similar numbers of files and file sizes in each type of job).

**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 another job with the same priority to the same site. This time the job goes to the configured atlas:validation share and the DTRs have priority 64 (80 x 80 / 100). Note that the priority of a DTR only affects the its position within a share and does not influence the distribution of slots between shares.

**4.1.4 ARC Data Delivery Service Technical Description**

This page describes how to set up DTR over multiple hosts.
Introduction

Under the old staging system, it was possible to spread the transfer load over multiple hosts by deploying several A-REX services (or Grid Managers) and one GridFTP interface. The GridFTP interface randomly distributed jobs over the control directories of each A-REX. In DTR the split is done at a much lower-level - between the Scheduler and Delivery layers - so that the remote hosts only perform simple point-to-point transfers between physical endpoints and all the logic is kept in the main A-REX host. This has many advantages over the previous system:

• Keeping all the high-level logic in one place allows intelligent load balancing.
• The remote hosts do not do submission to the batch system and so all LRMS configuration only needs to be done in one place.
• A problem with a remote host does not mean a lost job - the transfer can simply be retried on another host.
• The setup and configuration of a remote host is very simple.

Several options for how to implement multi-host staging were considered, for example:

• Using the control dir for communication between the Scheduler on the main A-REX host and daemons on remote hosts, which launch delivery processes when required
• Submitting batch jobs from the Scheduler to perform transfers
• Running a service which accepts data transfer requests from the Scheduler and runs transfer processes

The last solution was chosen as being the cleanest design and easiest to implement due to the existing HED framework. This datadelivery service which implements the Delivery layer has been developed within HED and can be deployed on one or many remote hosts. Once a DTR has passed all the pre-processing steps and is ready for transfer, the Scheduler can send it to a remote datadelivery service to execute the transfer. The Scheduler will then poll the request until it has completed. The Scheduler spreads randomly the total number of transfers (up to the configured limit and depending on the configured access rights of each service) between all the staging hosts.

The architecture is shown in the figure below. The datadelivery service on the remote host acts as a very simple Scheduler which sends all transfer requests it receives straight to Delivery.

Installation

No extra components are necessary on the A-REX host.

For the remote staging hosts, the package nordugrid-arc-datadelivery-service can be found in the usual EPEL or NorduGrid repositories, or is built automatically when building the source tree. This package should be installed on each remote host. It depends on some core ARC packages and the usual external packages that an ARC installation requires.

IMPORTANT If support for transfer protocols that are not included in the nordugrid-arc-plugins-base package (e.g. GridFTP or Xrootd) is required then the corresponding plugins packages must also be installed.

CA certificates are required on each remote host to authenticate connections with storage services. Depending on the local set up, each host may also require a host certificate (more info below).

To start/stop the service:

/etc/init.d/arc-datadelivery-service start|stop (as superuser)

The log file for the service can be found at /var/log/arc/datadelivery-service.log (configurable - see below). If logrotate is running, this log will be rotated every day.

Configuration

Remote Hosts

The datadelivery service uses the [datadelivery-service] section of the arc.conf configuration file.
Fig. 4.10: Multi-host datastaging
By default the datadelivery service runs with TLS enabled and a host certificate is required for each host. The path to the host credentials may be specified by the usual x509 options. Host certificates are not a strict requirement and it is possible to run without TLS using the secure=no option. This means that A-REX can no longer verify the authenticity of the remote hosts and so the decision to run with or without host certificates should be based on local site policy.

The following configuration options are supported:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x509_host_key</td>
<td>Path to host key</td>
<td>/etc/grid-security/hostkey.pem</td>
</tr>
<tr>
<td>x509_host_cert</td>
<td>Path to host cert</td>
<td>/etc/grid-security/hostcert.pem</td>
</tr>
<tr>
<td>x509_cert_dir</td>
<td>Path to CA certificates</td>
<td>/etc/grid-security/certificates</td>
</tr>
<tr>
<td>hostname</td>
<td>Hostname of service host</td>
<td>localhost</td>
</tr>
<tr>
<td>port</td>
<td>Port on which service runs</td>
<td>443</td>
</tr>
<tr>
<td>pidfile</td>
<td>pid file</td>
<td>/var/run/arched-datadelivery-service.pid</td>
</tr>
<tr>
<td>logfile</td>
<td>Log file</td>
<td>/var/log/arched-datadelivery-service.log</td>
</tr>
<tr>
<td>loglevel</td>
<td>Logging level (0 (FATAL) to 5 (DEBUG))</td>
<td>2 (WARNING)</td>
</tr>
<tr>
<td>user</td>
<td>User under which service runs (should only be changed in special cases)</td>
<td>root</td>
</tr>
<tr>
<td>secure</td>
<td>Set to &quot;no&quot; if the service should run without a host certificate</td>
<td>yes</td>
</tr>
<tr>
<td>allowed_ip</td>
<td>IP address authorized to access service (can be specified multiple times)</td>
<td>No default, must be specified</td>
</tr>
<tr>
<td>allowed_dn</td>
<td>DN authorized to access service (can be specified multiple times)</td>
<td>No default</td>
</tr>
<tr>
<td>transfer_dir</td>
<td>Path the service is allowed to read/write to (can be specified multiple times)</td>
<td>No default, must be specified must be specified</td>
</tr>
</tbody>
</table>

At least one allowed_ip and at least one transfer_dir are the only mandatory parameters, but it is can be useful to change interface, port and loglevel from the default.

Since the service can copy files to and from the service host, it is dangerous to allow open access to any clients. Usually allowed_ip is set to the IP address of the A-REX host since this is the only host which should have access to the service. The service can be further locked down by specifying authorized DNs so that only certain users are allowed to have their jobs’ files staged by the service. Note that DN filtering is not possible if secure=no is specified.

It is also dangerous to allow requests which can copy to or from anywhere on the host filesystem, so filesystem access is restricted through the transfer_dir parameter, which specifies the path(s) that requests are allowed to use. A transfer_dir should be specified for every cache and session dir. In some situations it may be desirable to set for example one cache per remote host where the cache is local to the host. A-REX checks on start-up which dirs are accessible by which remote hosts and uses that info to direct the DTRs to the right hosts.

Configuration example:

```
[datadelivery-service]
loglevel = 3
hostname = delivery.host.1
port = 60002
allowed_ip = 1.2.3.4
transfer_dir = /var/arc/session
transfer_dir = /var/arc/cache
```

It is vital that the cache and session file systems are mounted on the same path on the A-REX and each remote host, as the system assumes that a transfer to the cache or session directory can be done using the same path on all hosts. It is also vital that user accounts are in sync across all hosts, so that a user represented by a uid on the A-REX host maps to the same account on all the remote hosts. No mapping of DN to local user is done on the remote hosts, and so no mapping infrastructure like gridmap files is required. The local user id of the user owning
the transfer is passed in the request to the service and the transfer process is executed under that uid (except when writing to cache, when the root account is used).

The service is started and stopped by the init script arc-datadelivery-service:

```
$ARC_LOCATION/etc/init.d/arc-datadelivery-service start
```

**A-REX Host**

The configuration for setting up A-REX to use remote datadelivery services uses options in the [arex/data-staging] section of the regular arc.conf.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>delivery-service</td>
<td>URL of remote host which can perform data delivery. Hostname and port must match those specified in the datadelivery service configuration.</td>
<td>None</td>
</tr>
<tr>
<td>localdelivery</td>
<td>Whether local delivery should also be done</td>
<td>no</td>
</tr>
<tr>
<td>remotesizelimit</td>
<td>File size limit (in bytes) below which local transfer is always used</td>
<td>0</td>
</tr>
<tr>
<td>usehostcert</td>
<td>Whether the host certificate should be used in communication with remote datadelivery services instead of the user’s proxy</td>
<td>no</td>
</tr>
</tbody>
</table>

Example:

```
[arex/data-staging]
deliveryservice = https://delivery.host.1:60002/datadeliveryservice
deliveryservice = https://delivery.host.2:60002/datadeliveryservice
localdelivery = yes
remotesizelimit = 1000000
```

Multiple remote datadelivery services can be specified, and the Scheduler will randomly divide DTRs between those services where the transfer is allowed (according to allowed_dir configuration on each service). The presence of a datadeliveryservice option turns off the regular “local” delivery on the A-REX host so only the remote service will be used. If it is desired to also do transfers on the A-REX host, `localdelivery = yes` must be used. If no remote services are specified then local delivery is always enabled.

If the datadelivery service is running with secure=no then https should be replaced by http in the deliveryservice URLs.

Normally the credentials of the user who submitted the job are used for communication between A-REX and remote datadelivery services, however for extra security it is possible to use the A-REX host certificate for this instead, by specifying `usehostcert = yes`. In this case the host cert is only used for establishing the secure connection with the remote service, it is still the user’s credentials which are delegated and used for the transfer. The host cert must also be able to be used as a client certificate, in other words must have the X509 extension “X509v3 Extended Key Usage: TLS Web Client Authentication”. This option has no effect if the datadelivery service is running with secure=no.

Communication with remote services involves some degree of overhead such as the SSL handshake, delegating credentials etc, and when transferring small files this overhead can become a significant fraction of the transfer time. Therefore it is possible to specify a file size limit (in bytes) with `remotesizelimit`. Any files smaller than this limit will use local transfer, even if local transfer is disabled through `localdelivery = no`.

**Deployment Scenarios**

Several ways of deploying multi-host data staging are possible, using any number of remote hosts. Two examples are shown here.
Shared Storage

The cache and session directories are on a storage system such as Lustre or GPFS, which is mounted on all hosts. All hosts can access the cache and session directory and so DTRs will be split randomly between them. No transfer is done on the A-REX host.

Local Caches

Each remote host has its own local disk as a cache and these caches are mounted on the A-REX host. DTRs will be sent to the host corresponding to the cache chosen for the DTR so that all cache transfers are to local disks. The session directory is not available on the remote hosts so all uploads and non-cache downloads will run on the A-REX host.

Security

The remote datadelivery service may require a host certificate (see above) and allow incoming connections from the A-REX host, but no other incoming connections are required. It must have outbound connectivity to the world to perform the transfers. As explained above, access is normally restricted to the A-REX host and transfers restricted to the cache and session directories. For each transfer A-REX delegates the credentials of the user who submitted the job to the remote service, which creates a temporary file containing the credentials. This file is deleted when the transfer finishes. Creating a file should become unnecessary if pure in-memory credentials are fully supported by ARC and all transfer protocols.
The transfer process itself is executed under the uid of the session directory owner, unless the transfer is to cache in which case it is executed by root. It is therefore important that user accounts are synchronised across all hosts. The file systems with the cache and session directories must be mounted on the same paths with the same user access rights on all hosts.

**Proxies**

HED services do not support legacy proxies such as those generated by default by voms-proxy-init. In order to use remote datadelivery services, the jobs must be submitted to A-REX using RFC proxies, which can be generated by arcproxy or giving the option \texttt{-rfc} to voms-proxy-init. If a legacy proxy is used, local transfer will be used even if it is disabled in the configuration.

**Monitoring Remote Hosts**

When the first DTR is received by the Scheduler, it pings all the configured remote datadelivery services to check that they are running and to get the list of allowed directories from each one. If a service is unreachable it will not be used. If all services are unreachable then local delivery will be used even if it is turned off in the configuration. The allowed directories information is used to direct DTRs to services where the transfer is allowed. This procedure is repeated every 5 minutes (for the first DTR received after the 5 minute limit), and only the successful services are used until the next check (unless none are successful, in which case the check is done for every DTR until one succeeds). This means that configuration changes in the remote hosts will be picked up automatically after some time. However any changes in A-REX’s configuration, such as adding a new remote host, require an A-REX restart to be effective.

If remote datadelivery services are enabled, the number of DTRs assigned to each one can be seen on separate plots in Gangliarc.
4.1.5 The ARC Cache Index (ACIX)

The ARC Cache Index (ACIX) is a catalog of locations of cached files.

It consists of two components, one on the computing resource: the ACIX Scanner, and the ACIX Index which indexes the cache locations retrieved from the ACIX Scanners. These components can be found respectively in the packages nordugrid-arc-acix-scanner and nordugrid-arc-acix-index. They both depend on a third package, nordugrid-arc-acix-core.

ACIX Scanner

The ACIX Scanner periodically scans the A-REX cache and constructs a Bloom filter of cache content. This filter is a way of representing the cache content in an extremely compressed format, which allows fast query of any element of the filter and efficient upload of the content to an index server.

This type of compression however has the possibility of giving false-positives, i.e. a certain file may appear to be present in the cache according to the filter when it is not. The ACIX Scanner runs in an HTTPS server and the filter is accessible at the endpoint https://hostname:5443/data/cache.

It scans the caches specified in the A-REX arc.conf. It does not require any configuration but some options can be changed and it is important to make sure the ACIX Scanner port (default 5443) is open in the firewall.

ACIX Index

The ACIX Index server runs independently of the ACIX Scanner and A-REX, but can be deployed on the same host as both of them. It is configured with a list of ACIX Scanners and periodically pulls the cache filter from each one. It runs within an HTTPS server through which users can query the cached locations of files. Configuration uses the regular arc.conf file in the [acix-index] block. Here ACIX Scanners are specified by the cachescanner option. For example:

```
[acix-index]
cachescanner = https://my.host:5443/data/cache
cachescanner = https://another.host:5443/data/cache
```

The ACIX Index server can be queried at the endpoint https://hostname:6443/data/index and the list of URLs to check are given as comma-separated values to the option “url” of this URL, e.g:

```
```

A JSON-formatted response is returned, consisting of a dictionary mapping each URL to a list of locations. If remote access to cache is configured as described above then the location will be the endpoint at which to access the cached file, for example https://a-rex.host/a-rex/cache. If not then simply the hostname will be returned.

Using ACIX with A-REX Data Staging

ACIX can be used as a fallback mechanism for A-REX downloads of input files required by jobs by specifying use_remote_acix in the [arex/data-staging] block of arc.conf, e.g.:

```
[arex/data-staging]
use_remote_acix = https://cacheindex.ndgf.org:6443/data/index
```

If a download from the primary source fails, A-REX can try to use any cached locations provided in ACIX if the cache is exposed at those locations. In some cases it may even be preferred to download from a close cache rather than Grid storage and this can be configured using the preferredpattern configuration option which tells A-REX in which order to try and download replicas of a file.
Using ACIX for ARC client brokering

ACIX can also be used for data-based brokering for ARC jobs. An ACIX-based broker plugin written in Python comes packaged with the ARC client tools (in $ARC_LOCATION/share/arc/examples/PythonBroker/ACIXBroker.py) and can be used for example with:

```
[user ~] $ arcsub -b PythonBroker:ACIXBroker.ACIXBroker:https://cacheindex.ndgf.org:6443/data/index
```

Target sites for job submission are ranked in order of how many input files required by the job are cached there. See the comments inside this Python file for more information.

Deployment use-case

![ACIX deployment scenario diagram](image)

Fig. 4.13: ACIX deployment scenario, with one global ACIX Index and a local ACIX Index for CE 1a and CE 1b.

Fig. 4.13 shows an example ACIX set up. Each CE runs a ACIX Scanner and there is a central ACIX Index server which pulls content from all CEs. In addition there is one site with two CEs, CE 1a and CE 1b.

In order to do data-based brokering on just those two sites (and ease the load on the global ACIX Index server), a local ACIX Index is running which pulls content from only these two sites. In such a setup it may be desired to prefer to download data from the cache on CA 1a to CE 1b and vice versa, so those CEs could be configured with the Local ACIX Index server as the `use_remote_acix` and each other’s hostname first in `preferredpattern`.

4.1.6 CandyPond technical description

This page describes technical details of CandyPond, which stands for “Cache and deliver your pilot on-demand data”.

212 Chapter 4. Technical Documents Describing ARC Components
Description and Purpose

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. CandyPond exposes various operations of the cache to the outside and can be useful in a pilot job model where input data for jobs is not known until the job is running on the worker node. When the pilot picks its payload it can contact CandyPond to gain access to a file that is already cached, or if it is not cached ask CandyPond to download the file to cache.

Installation and Configuration

CandyPond is an integral part of A-REX and is available as part of the nordugrid-arc-arex package. It is enabled in A-REX by adding the block [arex/ws/candypond] to arc.conf. The [arex], [arex/ws] and [arex/data-staging] blocks are also required.

Runtime Environment Configuration

A runtime environment ENV/CANDYPOND exists to provide a convenient python module arccandypond to the job running on the worker node. This can either be used as a command line interface or as a python API. Note that the ENV/PROXY runtime environment is also needed in order to have access to the proxy on the worker node.

ENV/CANDYPOND will automatically detect the correct URL of the CandyPond service, but if it is desired to use a different URL then it can be set with

```
arccntl rte params-set ENV/CANDYPOND CANDYPOND_URL <url>
```

Command Line Interface

```
arccandypond get <url> <file>
```
can be used in place of whatever usual command the job would use to download input data. This command asks Candypond to download the url to cache if not already present, and link to the file specified in the job’s working directory.

```
arccandypond check <url>
```
can be used to check if the given url already exists in the cache. It will exit with 0 if the file is present, 1 if not, or 2 if an error occurred.

Python API

The job can import the module and use the CacheLink and CacheCheck methods to perform the equivalent of get and check commands.

Example Use Case

In this example a job is submitted which uses arccandypond to download input data to cache and have it available to the job.

The xrsf file defines the required runtime environments. Note that no input files are specified.

```
$ cat candypond.xrsl
6
("executable" = "candypond.sh")
("runtimeenvironment" = "ENV/CANDYPOND")
("runtimeenvironment" = "ENV/PROXY")
("jobname" = "candypond_test")
("walltime" = "3600")
```

(continues on next page)
The job script uses candypond to download the input file to cache and link to the job’s working directory:

```
#!/bin/sh
arccandypond get http://www.nordugrid.org:80/data/run.sh run.sh
ls -lrt
echo
cat run.sh
```

Submit the job:

```
$ arcsub candypond.xrsl
Job submitted with jobid: https://...
```

Check the output:

```
$ arccat https://...
('http://www.nordugrid.org:80/data/run.sh': {'0', 'Success'})
total 28
-rwx------- 1 dcameron dcameron 257 Apr 10 20:23 candypond.sh
drwx------ 4 dcameron dcameron 4096 Apr 10 20:23 arc
-rw------- 1 dcameron dcameron 10662 Apr 10 20:23 user.proxy
-rw------- 1 dcameron dcameron  0 Apr 10 20:23 stderr
lrwxrwxrwx 1 dcameron dcameron  89 Apr 10 20:23 run.sh -> /opt/var/arc/cache/
˓→joblinks/eOGODmV3WyunPSAtDmVmuSEmABFKDmABFKDmJSFKDmGBFKDm5lgA5m/run.sh
-rw------- 1 dcameron dcameron  62 Apr 10 20:23 stdout

#!/bin/sh
GCC=`which g++ 2>/dev/null`
if [ -z $GCC ]; then
    echo "Could not find the g++-compiler!"
    exit 0
fi
make
chmod 755 prime
./prime $1
```

Note the symbolic link to the cache.

**Issues and Notes**

- Calls to `arccandypond get` may block for a long time if the file needs to be downloaded to cache and A-REX is already busy with data staging or the file is very large. A timeout option will be added in the future.
- CandyPond (like the A-REX web service interface it is part of) does not accept legacy proxies. This type of proxy is created by default with older versions of grid/voms-proxy-init, but an RFC-compliant proxy can be generated using the -rfc option.
• CandyPond links files to the session dir. If a scratch directory is used for executing the job, the cache files are moved there from the session directory. This requires that the scratch dir is accessible from the CandyPond host, so it cannot be used in situations where the scratch directory can only be accessed by the underlying LRMS.

4.2 JURA Accounting Technical Details

General accounting configuration and operations flows are described in Accounting with JURA. This section contains more technical details about implementation of each component of accounting subsystem.

4.2.1 Records processing and publishing

![Diagram of ARC Computing Element and SGAS](image)

**Fig. 4.14: ARC CE accounting: records creation, processing and publishing**

**AREX Accounting Records (AAR job log files)**

The **A-REX Accounting Records (AAR)** are job log files generated by A-REX. AAR is the only source of accounting information for JURA. AARs are written by A-REX based on job data available, including .diag files that backend scripts creates based on batch system data and/or GNU time utility measurements. These job log files reside under the `<control_dir>/logs` directory. The name of the AAR 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 AAR job log file consists of `name=value` lines, where `value` is either a job-related resource consumption data or a static info like name, ID or proxy certificate.

A-REX generates at least two job log files for each job: one at the time of job submission, another one after the job finishes, and possibly others at various job events. Please note JURA makes use only one of the A-REX generated files belonging to the same job: the one that corresponds to the FINISHED job event (such state is indicated by the `status={completed|failed|aborted}`).

**JURA initial AAR processing**

A-REX periodically runs `jura` that loop over available A-REX job log records in the `<control_dir>/logs`. JURA opens all the files and processes only those that corresponds to a FINISHED job state.
JURA converts AARs to *per-job per-destination* extended AARs that contains the target information from `arc.conf` as well. One extended AAR is generated per accounting target. The extended AAR job log files named `<jobid>_<random>_<random2>` where first `<random>` is taken from original AAR.

The original AAR `<jobid>_<random>` file is deleted once *per-destination* extended AAR logs are created by JURA.

**Note:** JURA as part of the initial processing deletes all files corresponding to non-finished job states.

### JURA Publishing loop

JURA publishing subsystem loop over *extended AAR* logs in the A-REX `<control_dir>/logs` directory. JURA generates records in the Usage Record (UR) format proposed by the Open Grid Forum (OGF) for SGAS or Compute Accounting Record (CAR) XML for APEL.

The *extended AAR* job log file is deleted once record is successfully submitted, thus preventing multiple insertion of same usage records. If submission to destination fails, the extended AAR 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.

Please note that the JURA publishing loop is backward compatible with ARC 5 implementation.

### Reporting to SGAS

SGAS has a simple custom web service interface loosely based on WS-ResourceProperties. JURA uses the insertion method of this interface to report URs directly using ARC HTTP client implementation. The corresponding processed extended AAR job log files are deleted after receiving a non-fault response from the service.

To increase communication efficiency JURA sends URs in batches. SGAS accepts a batch of URs in a single request. The batch is an XML element called `UsageRecords`, containing elements representing URs.

The process of handling batches is the following: JURA does not send all usage records immediately after generation, but instead collects them in a batch until reaching the maximal number of records or until running out of job log files. The maximal number of URs in a batch can be set as a `urbatchsize` configuration parameter of SGAS target.

### Reporting to APEL

APEL uses the SSM framework for communication.

JURA send records to APEL by means of invoking helper `ssmsend` process that uses SSM python libraries developed by APEL.

ARC ships minimal set of SSM libraries along with A-REX binary packages to allow SSM usage. If SSM binary packages from APEL are available for your OS (e.g. EL6), you can install this packages and they will be used instead of those shipped with ARC automatically.

JURA prepares the messages to be sent by `ssmsend` and puts them info `SSM Outgoing` directory located in the `/var/spool/arc/ssm/<destination hostname>/outgoing/00000000/`. Generated messages are XML based CAR records with file name format `<YYYYMMDDhhmss>`.

Reporting to APEL also works with sending records in batches. The default `urbatchsize` value is set to 1000 according to APEL recommendations.

### 4.2.2 Accounting archive

TODO: `jura-archive-manager`
4.2.3 Republishing process

TODO: Describe republishing workflow in bit more details

4.2.4 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 `x509_set of confiurartion options of arc.conf`. No proxies are used for communication with accounting services.

The only access restriction made by a SGAS service is matching the Distinguished Name of the client (in this context JURA) with a set of trusted DNs. When access is granted, policies are then applied by SGAS, allowing either publishing and/or querying rights. Clients with publishing right can insert any UR, regardless of content. By default, querying right only allows retrieving URs pertaining to jobs submitted by the querying entity.

4.2.5 Implementation and API

JURA as part of the ARC software stack is written in C++, and utilizes the functionality provided by the ARC libraries, including secure HTTPS communication provided by the ARC plugable TLS and HTTP modules.

The modular design is also present in the usage reporting part of the JURA code, making it possible to extend JURAs support of accounting services. To create a JURA module one should simply write a C++ class which inherits from the abstract `Arc::Destination` class, and it must extend the two methods:

- `static Arc::Destination* Arc::Destination::createDestination(Arc::JobLogFile&)`
- `void Arc::Destination::report(Arc::JobLogFile&)`
The static `createDestination` method should initialize an object of the specialized class, using the configuration options specified in the passed `Arc::LogFile` object, and the memory allocated by the method should be freed by the caller. Then the `report` method should carry out the transfer of the UR, represented by the `JobLogfile` object, to the accounting service.

JURA archive manager is written in Python and share the classes for managing ARC components with ARC Control Tool.

### 4.2.6 Limitations

In the following list some issues which limits the functionality of JURA is described:

- The current implementation of JURA and A-REX supports only one expiration time for all the reporting destinations. Even though the configuration enables the specification of different expiration values per reporting destination, it is not taken into account by the system, the last value is used as the common expiration time value.

- It is not possible to use different credentials per destinations.

- If you are updating from ARC5 with an old jura accounting archive already containing records, the conversion process to index the archive structure will be initiated and will cause serious system load until finished. To avoid old archive conversion, you can move records before update.

- Some optional UR properties are not supported.

- Memory can be reported incorrectly with buggy GNU “time” results.
### 4.2.7 Definition of the A-REX Accounting Record including JURA attribute mappings to SGAS and APEL

#### Table 4.2: Attributes used in current implementation

<table>
<thead>
<tr>
<th>A-REX Accounting Record (AAR)</th>
<th>SGAS OGF-UR</th>
<th>APEL CAR</th>
<th>Content description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ngjobid</td>
<td>RecordIdentity is composed of ngjobid and hostname taken from the headnode.</td>
<td>RecordIdentity is composed of ngjobid and hostname taken from the headnode.</td>
<td>The global unique jobid assigned by AREX.</td>
</tr>
<tr>
<td>globalid</td>
<td>JobIdentity. GlobalJobId</td>
<td>JobIdentity. GlobalJobId</td>
<td>The global unique jobid assigned by AREX. For gridftp interface it contains full URL.</td>
</tr>
<tr>
<td>localid</td>
<td>JobIdentity. LocalJobId</td>
<td>JobIdentity. LocalJobId</td>
<td>LRMS job ID</td>
</tr>
<tr>
<td>jobname</td>
<td>JobName</td>
<td>JobName</td>
<td>User specified job name</td>
</tr>
<tr>
<td>headnode</td>
<td>MachineName</td>
<td>MachineName, SubmitHost, Site</td>
<td>The A-REX job submission endpoint URL used for this job</td>
</tr>
<tr>
<td>lrms</td>
<td>not used</td>
<td>Infrastructure (used as a part of it)</td>
<td>The LRMS behind A-REX</td>
</tr>
<tr>
<td>queue</td>
<td>Queue</td>
<td>Queue</td>
<td>The name of the LRMS queue of the job</td>
</tr>
<tr>
<td>nodename</td>
<td>Host</td>
<td>Host</td>
<td>WN name(s) as given by LRMS separated by :</td>
</tr>
<tr>
<td>clienthost</td>
<td>SubmitHost (port removed)</td>
<td>not used</td>
<td>Client connection socket from the client to A-REX</td>
</tr>
<tr>
<td>usersn</td>
<td>UserIdentity. GlobalUserName</td>
<td>UserIdentity. GlobalUserName</td>
<td>The global user identity, at the moment it is the SN from the certificate</td>
</tr>
<tr>
<td>localuser</td>
<td>UserIdentity. LocalUserId</td>
<td>UserIdentity. LocalUserId</td>
<td>The mapped local userid</td>
</tr>
<tr>
<td>usercert</td>
<td>UserIdentity.VO and child structures</td>
<td>UserIdentity. Group and UserIdentity. GroupAttribute</td>
<td>contains the full proxy cert chain</td>
</tr>
<tr>
<td>projectname</td>
<td>ProjectName</td>
<td>UserIdentity. GroupAttribute</td>
<td>User-defined name of the project the job belongs to</td>
</tr>
<tr>
<td>status</td>
<td>Status</td>
<td>Status</td>
<td>The terminal state of an A-REX job: aborted, failed, completed</td>
</tr>
<tr>
<td>exitcode</td>
<td>not used</td>
<td>ExitStatus</td>
<td>The exit code of the payload in the LRMS</td>
</tr>
<tr>
<td>submissiontime</td>
<td>StartTime</td>
<td>StartTime</td>
<td>The timestamp of job acceptance at A-REX</td>
</tr>
<tr>
<td>endtime</td>
<td>EndTime</td>
<td>EndTime</td>
<td>The timestamp when the job reached the terminal state in A-REX</td>
</tr>
<tr>
<td>nodecount</td>
<td>NodeCount</td>
<td>NodeCount</td>
<td>Number of allocated worker nodes</td>
</tr>
<tr>
<td>inputfile</td>
<td>FileTransfers</td>
<td>not used</td>
<td>Details of downloaded inputfile: url, size, transfer start, transfer end, downloaded from cache</td>
</tr>
<tr>
<td>outputfile</td>
<td>FileTransfers</td>
<td>not used</td>
<td>Details of uploaded outputfile: url, size, transfer start, transfer end</td>
</tr>
<tr>
<td>usedmemory</td>
<td>Memory</td>
<td>Memory</td>
<td>Memory used by the job</td>
</tr>
<tr>
<td>usedmaxresident</td>
<td>Memory</td>
<td>Memory</td>
<td>Maximum resident memory used by the job</td>
</tr>
<tr>
<td>usedavageresident</td>
<td>Memory</td>
<td>Memory</td>
<td>To be dropped from the AAR schema</td>
</tr>
</tbody>
</table>
### Table 4.3: Proposed NEW attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cores</td>
<td>The number of cores allocated to the job</td>
</tr>
<tr>
<td>systemsoftware</td>
<td>The type and version of the system software (i.e. opsys, glibc, compiler, or the entire container wrapping the system software)</td>
</tr>
<tr>
<td>WNinstance</td>
<td>Coarse-grain characterization tag for the WorkerNode, e.g. BigMemory or t2.micro (aka Amazon instance type)</td>
</tr>
<tr>
<td>RTEs</td>
<td>List of used RTEs, including default ones as well.</td>
</tr>
<tr>
<td>data-stagein-volume</td>
<td>The total volume of downloaded job input data in GBs</td>
</tr>
<tr>
<td>data-stagein-time</td>
<td>The time spent by the DTR system to download input data for the job</td>
</tr>
<tr>
<td>data-stageout-volume</td>
<td>The total volume of uploaded job output data in GBs</td>
</tr>
<tr>
<td>data-stageout-time</td>
<td>The time spent by the DTR system to upload output data of the job</td>
</tr>
<tr>
<td>lrms-submission-time</td>
<td>The timestamp when the job was handed over to the LRMS system</td>
</tr>
<tr>
<td>lrmsstarttime</td>
<td>The timestamp when the payload starts in the LRMS</td>
</tr>
<tr>
<td>lrmsendtime</td>
<td>The timestamp when the payload completed in the LRMS</td>
</tr>
<tr>
<td>authtokenattributes</td>
<td>to be implemented to store e.g. VO attributes and simillar capabilities</td>
</tr>
<tr>
<td>submissioninterface</td>
<td>to be implemented, at the same time clean up globalid that depends on submissioninterface type</td>
</tr>
<tr>
<td>benchmark</td>
<td>The type and the corresponding benchmark value of the assigned WN</td>
</tr>
<tr>
<td>usedscratchspace</td>
<td>The used size of scratch dir at the end of the job termination in the LRMS.</td>
</tr>
</tbody>
</table>
Table 4.4: NOT USED SGAS or APEL attributes

<table>
<thead>
<tr>
<th>SGAS OGF-UR</th>
<th>APEL CAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessID</td>
<td></td>
</tr>
<tr>
<td>Charge</td>
<td>Charge</td>
</tr>
<tr>
<td>Swap</td>
<td>Swap</td>
</tr>
</tbody>
</table>

4.3 Old Relevant Technical Documents

Note: Many of the technical documents exist for ARC5 only. Those that are verified to be relevant for ARC6 will be listed below

4.3.1 Hosting Environment of the Advanced Resource Connector middleware

Document gives a deep technical description of the HED service container.
4.3.2 A Client Library for ARC

Document describes from a technical viewpoint the plugin-based client library of ARC.
If you are looking for development internal details of ARC (like how some stuff was coded) this part of documentation is for you. Mainly for those who want to contribute to the project development, advanced troubleshooters or just interested.

5.1 Implementation Details for Developers

5.1.1 General arc.conf python configuration parser

Parsing configuration

Initial configuration parsing

The latest version of arcconfig-parser designed to operate with defaults file that holds default values for all possible parameters.

At the time of initial parsing the following chain is executed:

- All blocks and options are parsed from arc.conf
- For all blocks defined in arc.conf missing options are added from defaults file
- Special constructs in values are substituted (see Special constructs can be used as values)

Optional by design parameters that does not have default value (specified with not set value) are not included to the parsed configuration.

Runtime configuration

Configuration that includes both arc.conf and defaults config called runtime configuration.

In some cases it is useful to save and load runtime configuration:

- To supply C++ services (a-rex, gridftp) with configuration that includes defaults from common place
- For repetitive operations on config to eliminate full-chain processing of each invocation

To save runtime configuration to the default location (/var/run/arc):

```
arcconfig-parser --save
```

To save runtime configuration to specified location:

```
arcconfig-parser --save -r /var/run/arc/arex.arc.conf
```

To load runtime configuration instead of full-chain processing and e.g. get the value of x509_host_key in [common] block:
Special constructs can be used as values

Defaults includes references to another config parts to be consistent with the implied arc.conf structure.

The most obvious example is if x509_host_key not found in e.g. [arex/jura] block it should be taken from [common].

Config parser is following this logic (especially in respect to defaults) and use special constructs to accomplish this behaviour.

Command substitutions

Configuration option values can contain the construct $EXEC{<command>}$ that substituted to the stdout of <command>.

For example:

```
hostname=$EXEC{hostname -f}
```

Option values substitutions

The construct $VAR{[block]option}$ can be used to substitute the values of another option value.

If option is in the same block as referencing option block name can be omitted - $VAR{option}$.

For example:

```
x509_host_key=$VAR{[common]x509_host_key}
```

```
bdii_update_cmd=$VAR{bdii_location}/sbin/bdii-update
```

Evaluation of simple code

For limited number of cases arc.conf default values relies on arithmetic operations. For this purpose the $EVAL{string}$ special construct had been introduced.

For example:

```
bdii_read_timeout=$EVAL{$VAR{bdii_provider_timeout} + $VAR{[arex]infoproviders_→timelimit} + $VAR{[arex]wakeupperiod}}
```

Getting the configuration values

If --option argument is passed to arcconfig-parser parser returns the value of the specified option to stdout.

Without --option arcconfig-parser can be used to operate with configuration blocks:

- check blocks existance (exit code used to indicate the status of the check)
- return the list of subblocks

With the --export option arcconfig-parser allows to export config in the following formats:

- json - returns entire configuration or subset of blocks as-is in JSON to stdout
• **bash** - for [common] block or specified configuration subset returns `CONFIG_option_name=value` pairs to stdout. Block names ARE NOT included in the exports and option values precedence will be used in the order of passed blocks. If automatic subblocks expansion used with bash export, for every block in sequence - it’s subblocks are processed first (in `arc.conf` defined order). It is possible to filter the options that will be exported with additional `--filter` option that can be specified several times.
Common configuration parsing sequence

Developers entry-point to put info

arc.conf.reference  ➔  Substitution syntax

Binary Distribution

DOMContentLoaded

//etc/arc.conf  ➔  /usr/share/doc

/usr/share/doc  ➔  arc.parser.defaults

arc.parser.defaults  ➔  buildtime

1. parse, get defined blocks

2. add defaults for defined blocks

3. evaluate substitutions

runtime configuration

dump config

export JSON  ➔  export BASH

get value

/startup scripts

/startup scripts

/get value

/var/run/arc/arc.conf

define ENV variables

/startup scripts

/startup scripts

start a-rex

Examples

Get value of option in block:
Get value of option in blocks in order they are specified (e.g. if not found in [gridftpd] look in the [common] block):

```
# arcconfig-parser --block gridftpd --block common --option x509_host_key
/etc/grid-security/hostkey.pem
```

Export entire configuration to JSON:

```
# arcconfig-parser --export json
```

Export [infosys] block options to JSON (for Perl):

```
# arcconfig-parser --block infosys --export json
{
  "infosys": {"loglevel": "5"},
...
```

Export [infosys] block and all their subblocks options to JSON:

```
# arcconfig-parser --block infosys --subblocks --export json
("infosys/glue2/ldap": {"showactivities": "no"},
...
```

Export for BASH (compatible with config representation in shell-based LRMS backends):

```
# arcconfig-parser --block infosys --block arex --block common --export bash
CONFIG_controldir="/var/spool/arc/jobstatus"
CONFIG_defaultttl="1210000"
CONFIG_delegationdb="sqlite"
CONFIG_hostname="ce01.example.org"
CONFIG_maxrerun="5"
CONFIG_maxjobs="10000 -1"
CONFIG_runtimedir="/home/grid/arc/runtime"
CONFIG_sessiondir="__array__" # <= NEW define for multivalued values that indicate
˓
→ indexed vars
CONFIG_sessiondir_0="/mnt/scratch/grid/arc/session"
CONFIG_sessiondir_1="/home/grid/arc/session drain"
...
```

Export for BASH with exported options filtering:

```
# arcconfig-parser -b common -f hostname -f x509_cert_dir -e bash
CONFIG_hostname="ce01.example.org"
CONFIG_x509_cert_dir="/etc/grid-security/certificates"
```

Using BASH export:

```
# eval "$( arcconfig-parser --block infosys --block arex --block common --export
˓
→ bash )"
# echo "${CONFIG_gridmap}"
```

Check block(s) exists ([common/perflog] does not exists in the example):

```
# arcconfig-parser --block common/perflog --block arex
# echo $?
1
```

List block subblocks:

1 Block dependencies are now implied by defaults file, so for most cases it is enough to specify only block in question
2 HINT: use arcconfig-parser --export json | jq . to view highlighted JSON structure in shell

5.1. Implementation Details for Developers
Using parser as Python module:

```python
from arc.utils import config

# initial parsing with defaults
config.parse_arc_conf('/etc/arc.conf', '/usr/share/arc/arc.parser.defaults')

# get parsed dictionary and list of blocks in the arc.conf order
confdict = config.get_config_dict()
confblocks = config.get_config_blocks()

# get list of all [queue] subblocks sorted by name
sb = config.get_subblocks(['queue'], is_sorted=True)

# get value of 'x509_host_key' from [arex] block and than from [common]
if not found in [arex]
a = config.get_value('x509_host_key', ['arex', 'common'])

# get value of 'allowactivedata' option from [gridftpd] block
b = config.get_value('allowactivedata', 'gridftpd')

# get value of 'allowactivedata' option from [gridftpd] block (always return list)
c = config.get_value('allowactivedata', 'gridftpd', force_list=True)

d = config.get_value('allowactivedata', 'gridftpd', bool_yesno=True)

5.1.2 LRMS shell-backends overview for developers

CONFIG variables used in LRMS shell-backend:

lrms_common.sh:

```bash
$CONFIG_runtimedir [arex]
$CONFIG_shared_scratch [arex]
$CONFIG_shared_filesystem [arex]
$CONFIG_scratchdir [arex]
$CONFIG_gnu_time [lrms]
$CONFIG_nodename [lrms]
$CONFIG_enable_perflog_reporting [common]
$CONFIG_perflogdir [common]
```
submit_common.sh:

```
$CONFIG_defaultmemory [queue] [lrms]
$CONFIG_hostname [common]
$CONFIG_controldir [arex]
```

**lrms=boinc:**

```
$CONFIG_boinc_app_id [lrms]
$CONFIG_boinc_db_host [lrms]
$CONFIG_boinc_db_port [lrms]
$CONFIG_boinc_db_user [lrms]
$CONFIG_boinc_db_pass [lrms]
$CONFIG_boinc_db_name [lrms]
```

**lrms=condor**

```
# $CONFIG_enable_perflog_reporting [common] not in reference
# $CONFIG_perflogdir [common] not in reference
# $CONFIG_controldir [arex] (for perflog)

$CONFIG_condor_requirements [queue] [lrms]
$CONFIG_condor_rank [lrms]
# $CONFIG_shared_filesystem [arex]
$CONFIG_condor_bin_path [lrms]
$CONFIG_condor_config [lrms]
```

**lrms=fork:**

```
no variables
```

**lrms=ll:**

```
# $CONFIG_enable_perflog_reporting [common] not in reference
# $CONFIG_perflogdir [common] not in reference
# $CONFIG_controldir [arex] (for perflog)

$CONFIG_ll_bin_path [lrms]
$CONFIG_ll_consumable_resources [lrms]
$CONFIG_ll_parallel_single_jobs *not in reference
# $CONFIG_scratchdir [arex]
```

**lrms=lsf:**

```
# $CONFIG_enable_perflog_reporting [common] not in reference
# $CONFIG_perflogdir [common] not in reference
# $CONFIG_controldir [arex] (for perflog)

$CONFIG_lsf_architecture [lrms]
$CONFIG_lsf_bin_path [lrms]
```

**lrms=pbs:**

```
# $CONFIG_enable_perflog_reporting [common] not in reference
# $CONFIG_perflogdir [common] not in reference
# $CONFIG_controldir [arex] (for perflog)

$CONFIG_pbs_queue_node [queue]
$CONFIG_pbs_bin_path [lrms]
$CONFIG_nodememory [queue] ([infosys/cluster] parser_ ...

(continues on next page)
```

3 Here and following † prefix is for options and are used in *_common scripts and not unique to particular backend
|$CONFIG_pbs_log_path| lrms|
# $CONFIG_shared_filesystem| arex|

**lrms=sge:**

# $CONFIG_enable_perflog_reporting| [common] not in reference
# $CONFIG_perflogdir| [common] not in reference
# $CONFIG_controldir| [arex] (for perflog)

$CONFIG_sge_root| [lrms]
$CONFIG_sge_cell| [lrms]
$CONFIG_sge_qmaster_port| [lrms]
$CONFIG_sge_execd_port| [lrms]
$CONFIG_sge_bin_path| [lrms]
$CONFIG_sge_jobopts| [queue] [lrms]
# $CONFIG_scratchdir| [arex]

**lrms=slurm:**

# $CONFIG_enable_perflog_reporting| [common] not in reference
# $CONFIG_perflogdir| [common] not in reference
# $CONFIG_controldir| [arex] (for perflog)

$CONFIG_slurm_wakeupperiod| [lrms]
$CONFIG_slurm_use_sacct| [lrms]
$CONFIG_slurm_bin_path| [lrms]
# $CONFIG_shared_filesystem| [arex]

**Call graph**
Submitting jobs
Scanning jobs
Canceling jobs

Changes in ARC6 memory limits processing:

Current logic of memory limits processing:

- **nodememory** - advertise memory for matchmaking: max memory on the nodes (in [infosys/cluster] block or per-queue)
- **defaultmemory** - enforce during submission if no memory limit specified in the job description (in [lrms] block or per-queue)

The ARC6 logic is no enforcement = no limit

Backends behaviour with no memory enforcement limit:

- boinc - set to hardcoded 2GB
- condor - no enforcement
- form - no memory handling at all
- ll - no enforcement
- lsf - no enforcement
- pbs - no enforcement
- sge - no enforcement
- slurm - no enforcement

---

1. ARC5 logic was no enforcement = max node memory or 1GB if nodememory is not published (and not used for matchmaking)
2. exclusivenode is memory-based and nodememory value is used in this case
5.2 Contributing to Documentation

NorduGrid ARC6 documentation is mainly written in reStructuredText and built with Sphinx to HTML pages, LaTeX (for printable PDF versions) and ePub.

Recent source tree (master branch) build is available instantly via Coderefinery GitLab Pages and deployed to the nordugrid.org on nightly basis.

5.2.1 Commiting and reviewing changes

Contribution is possible by direct push to the repo, no need for merge requests if you are fixing typos or other content that does not requires review!

**Note:** In the rendered HTML version there is a link at the bottom of each page labeled *Edit page source on GitLab*. Using this link you can edit the page source in your web-browser without the need to checkout source tree.

In case you want to add something new that requires the review process:
- create the new branch (in the same repository, no need to fork)
- open a merge request to master branch
- in-line and general discussion of the particular contribution is inside merge request.

5.2.2 Documentation structure

As reflected in the main index page, the documentation logically divided into the following groups:

- Documentation for Infrastructure Users
- Documentation for Infrastructure Admins
- Documentation for Developers
- Technical Documents Describing ARC Components

Admins documentation is the most developed part for ARC6. We aimed to have dedicated documents for each particular topic and then bound them in the following way:

- The introductory *Try ARC6: towards distributed computing in a few minutes* that contains a zero-configuration case hands-on instruction to help new users and admins getting started with ARC CE.
- Main *ARC Computing Element Installation and Configuration Guide* that contains general installation and configuration flow for production Computing Element deployment. The guide itself should be brief enough and holds only main examples. All detailed instruction for each particular subsystem configuration should be written in the dedicated document that linked to this guide.
- *ARC Configuration Reference Document* automatically rendered from text version in the code source tree. Targeted for web-search. The ultimate configuration options description we have.
- All other dedicated documents goes to *Operating ARC CE Subsystems* section in case someone wants to access them directly instead of following installation guides links.

5.2.3 Source tree directory structure

Sources of documents in rST format are placed into the source directory that contains the following structure:

- **_static** - logo and CSS files

---

Do not touch unless you are modifying the Sphinx build itself. Nothing there affects documentation writing process.
• _templates - Sphinx HTML theme layout tuning
• _extensions - Custom sphinx extensions developed for ARC
• common - directory used to store common for all ARC customers documentation, including admins, users and developers.
  – repos - this directory holds repository configuration instructions for release, nightlies, etc. Main index page have repos pointers as well as Try ARC6 and Install Guide.
  – changelog - documents that describe changes between ARC versions.
• admins - directory used to store Documentation for Infrastructure Admins. The main documents that represent the entry points to ARC Computing Element installation and configuration resides directly in this directory.
  – details - this subdirectory designed to hold detailed configuration instructions for ARC CE subsystems, like RunTime Environments in ARC6 that describe all aspects of RTEs in-depth. This documents intentionally moved deeper in the main TOC tree to prevent first levels flooding. If you can find a chapter in main install guide that cat point to the in-depth document in question - this is a right place to store it.
  – commands - place for automatically generated command line option reference for ARC tools. At the time of writing the Python tools represented there.
  – archery - for the documents that not related to Computing Element another subdirectory should be created. At the time of writing only ARCHERY documentation of that kind is available.
• developers - directory used to store Documentation for Developers. As long as we don’t have many of this kind of documents everything that contains implementation details (how stuff coded, which variable used, etc) should go there.
• tech - directory used to store Technical Documents Describing ARC Components. Now it holds only index document that contains references to the available PDFs that is verified to be applicable to ARC6 release. To add more, use template inside the index file.
• users - directory used to store Documentation for Infrastructure Users. If document is user-oriented - place it here.
• sdk - place for Doxygen SDK documentation integration to Sphinx build
• testing - directory used to store the documents for ARC 6 Testing Area.
• wip - “work in progress” area hidden in the TOC tree. You can use it for incomplete documents that should not be publicly advertised yet, but still will be built and available via /wip/ URL in produced HTML files.

Index files

Each directory in the source tree contains index.rst file that is used to link other documents in the same TOC tree. Upper-level index.rst file contains references to index.rst files in the subdirectories, that in turns contains pointers to the other documents.

Note: When you add a new document, add a reference to the index.rst in the same directory

Just follow the index.rst chain (look for toctree keyword) starting from the source directory to get familiar with TOC linking structure.

Storing images

When you need to add images to your document you should upload the image file itself and refer to it from the .rst.

ARC6 documentation structure implies that image files are stored inside the images subdirectory in the document location. Then referencing is done by relative path, e.g:
5.2.4 Building the docs

The top directory in the source tree contains `build.sh` script that:

- checkout the ARC source code tree and configures it with packaging-like paths
- builds Doxygen SDK documentation
- converts `arc.conf.reference` to rST
- copies documentation parts from ARC source code tree (for `developers` section)
- prepare automatically generated documentation for CLI commands
- builds HTML
- builds PDF (with LaTeX)
- builds ePub

On commit the GitLab CI configured to automatically invoke the `build.sh` to produce rendered documentation. Documentation archive is available as CI job artifacts and for master branch is deployed to GitLab Pages.

To build the docs on your local machine you should have at least Sphinx installed. Additionally you should be able to configure ARC source tree and ARC Python command dependencies for auto-generated parts.

Complete and up-to-date list of dependencies defined for CI build and can be found in `.gitlab-ci.yml`.

PDF builds with LaTeX is the most heavy part from the both time and needed additional packages perspective. For HTML rendering local debugging it is recommended to use the `html` script argument the skips PDF and ePUB:

```
[user@localhost doc]$ ./build.sh html
```

Point your browser to `file:///path/to/doc/build/html/index.html` to view the HTML rendering locally.

5.2.5 Writing Documentation in reStructuredText

reStructuredText (reST) is the default plaintext markup language used by Sphinx. There is possible to render other markups but for consistency and better cross-referencing NorduGrid ARC6 documentation written solely in reStructuredText.

General Syntax

reStructuredText markup specification is well documented in the several sources and was designed to be a simple and readable in plain-text. Common text editors (including `vim` and `emacs`) recognize reST markup and provide syntax highlighting out of the box.

Start with reStructuredText Primer on Sphinx docs.

Complete reST Markup Syntax can be found on Docutils starting with Quick reStructuredText document.

Sphinx also uses interpreted text roles to insert semantic markup (cross-referencing, etc) into documents. To get familiar read this document.

Just study the markup following the documentation and reading the already written documents just here.
Code snippets

There are different options for representing the code, starting with the simple literal blocks identified with `::`

It is advised to use `code-block::` directive to enable syntax highlighted rendering:

```code-block:: ini
[gridftpd/jobs]
allowaccess = staticdn dnfromfile
```

In this example, the `ini` tag represents the syntax highlighting lexer identifier for Pygments. Look for available lexers to serve your needs. The most common for our docs are: `console`, `bash`, `ini` and `cfg`.

References

Referencing another parts of documentation is necessary to achieve usability. The typical referencing cases are:

Using custom label

Create label just above any of the paragraph headers with. The following markup creates `my_label` label.

```code-block::
.. _my_label:

My Heading
============
```

Refering to a label is possible from any other document with:

 read the :ref:`my_label`

In this case paragraph heading will be used for hyperlink text. If you want some custom text for hyperlink text use the following syntax:

 read this :ref:`text <my_label>`

Referring `arc.conf.reference`

Autogenerater reST rendering of `arc.conf.reference` already contains labels for all configuration options and blocks that can be used.

The label name has the following structure:

- `reference_<block name>[_<sub_block>_]<option name>` represents configuration option inside block
- `reference_<block name>[_<sub_block>_]` represents block itself

For example:

```
In the `arc.conf` there is a dedicated :ref:`[lrms] <reference_lrms>` block that defines the type of your LRMS.
*Job session directory* is configured with :ref:`reference_arex_sessiondir`
```

5.2. Contributing to Documentation
Referring bugz

Custom Sphinx plugin for ARC documentation introduces new reST roles that can be used to mention bugs in NorduGrid Bugzilla or GitLab Issues:

More details can be found in :bugz:`3802` and :issue:`57`.

Referencing docs

Referencing the whole document is similar to using labels, but instead of label name the document name (filename without extension) is used with :doc: keyword.

Example 1: Refer to the try_arc6.rst in the same source tree directory (relative path). Use the document header as hyperlink text:

:doc:`try_arc6`

Example 2: Refer to the repository.rst by absolute path (starting from sources top directory). Use the custom hyperlink text:

:doc:`NorduGrid Repositories </common/repos/repository>`

Adding notes

To highlight statement visually use notes and warnings:

```
.. note::
    Zero configured A-REX comes with EMI-ES and REST interfaces enabled.

.. warning::
    This information is valid for releases of ARC 6 starting from 6.0.
```

Adding images

In the ARC6 documentation images should be included with the caption. It is accomplished with the figure keyword.

After storing image inside images subdirectory, include it in reST document as:

```
.. figure:: images/shared_sessiondir_yes.svg
   :align: center
   :alt: Sessiondir is shared between ARC CE and WNs

   Sessiondir is shared between ARC CE and WNs. No local scratchdir defined.
```

Sphinx build had been configured with image format autocoversion feature. So you can use any image format, including vector graphics.

Graphviz

If you want to illustrate some process or structure that can be shown as graph, consider using built-in Graphviz functionality.

Simple usage examples can be found in admins/details/rtes.rst document within ARC documentation. Give it a try, it is easy.
Converting from other sources

Converting the old documentation parts from LaTeX or Mediawiki markdown to reST is not a magical process that do everything automatically unfortunately.

However you can get a good start with a pandoc tool that do the conversion.

E.g. to convert LaTeX source simply run:

```
[user ~]$ pandoc -f latex -t rst acix.tex > acix.rst
```

The amount of efforts to edit the resulted .rst file and fix formatting issues is completely depends on the source itself (e.g. how the console output was formatted in the origin document). Images and references in most cases should be fixed separately after the pandoc.
This part of the documentation targeted to distributed computing infrastructure users that use either clients or SDK to run jobs and handle data transfers.

6.1 CLI and SDK Documentation

6.1.1 ARC Client Installation Instructions

Note: There are no client installation instructions available in the new documentation format yet. Please use the old installation instructions for main points and hints.

6.1.2 ARC SDK Documentation
A dedicated area is created for the ARC6 testing and certification.

7.1 ARC 6 Testing Area

**Note:** This is the home for all ARC6 related testing activities, infrastructures and plans. Please add text, links, anything here

### 7.1.1 Sites with ARC 6 alpha or release candidate installed

Early installers of ARC 6 - Testing sites

<table>
<thead>
<tr>
<th>Site</th>
<th>OS</th>
<th>First installed</th>
<th>Current version</th>
<th>Mode</th>
<th>Issues reported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oslo Abel (ce01, ce02, ce03) (T1)</td>
<td>Centos 6</td>
<td>June 2018</td>
<td>Nightlies 201905290207</td>
<td>Production 17</td>
<td>BUGZ-3785, BUGZ-3737, BUGZ-3736, BUGZ-3781, BUGZ-3782, BUGZ-3780, BUGZ-3779, BUGZ-3735, BUGZ-3785, BUGZ-3802, BUGZ-3796.</td>
</tr>
<tr>
<td>Linkoping Bluegrass</td>
<td></td>
<td>October 2018</td>
<td>ARC RC 6</td>
<td>Production</td>
<td></td>
</tr>
<tr>
<td>Copenhagen T3</td>
<td>Centos 7</td>
<td>End-february</td>
<td>Nightlies: 201905070207</td>
<td>Production 15</td>
<td>Issue #57</td>
</tr>
<tr>
<td>Slovenia - SIGNET</td>
<td>Gentoo</td>
<td>March 2019</td>
<td>ARC RC 6</td>
<td>Production</td>
<td></td>
</tr>
<tr>
<td>Slovenia - SIGNET_NSC</td>
<td>Fedora 28</td>
<td>March 2019</td>
<td>Nightly installation equiv to RC6</td>
<td>Test</td>
<td></td>
</tr>
<tr>
<td>Ukraine - KNU</td>
<td>Centos 7</td>
<td>June 2018</td>
<td>ARC RC 5 (6?)</td>
<td>Production</td>
<td></td>
</tr>
</tbody>
</table>
List of bugs reported for ARC 6

Bugzilla search
Issues in GitLab

7.2 Changelogs/list of bugs

7.2.1 Main changes in ARC 6 compared to ARC 5

There are many improvements and changes in the new ARC release, and with these we hope to have greatly enhanced the ease of setting up, configuring and managing an ARC production site, and to have improved the reliability and scalability of the ARC CE by the internal restructuring ARC has undergone.

**Note:** Despite all the new features and code changes, the supported ARC 6 CE interfaces are unchanged - the latest ARC 5 clients are compatible with an ARC 6 CE and vice versa.

- **Complete overhaul of server-side ARC CE configuration:**
  - Configuration has been completely reworked, cleaned up, regrouped and made more streamlined.
  - In particular, the way authorization and user mapping is handled has been greatly improved. Worth mentioning is the support of higher granularity vo-based authorization, and that mapping via gridmap files is no longer required nor recommended.
  - ARC now comes with a so-called zero configuration - a preconfigured minimal ARC setup automatically installed with ARC, including test-CA and test-host certificate for immediate test submission.
  - Default values throughout ARC have had a big tidy up and are now handled consistently through the ARC runtime configuration.
  - Configuration blocks are now used to enable (turn on) or disable a functionality, a service or an interface. For example, the NorduGrid schema is no longer published unless it is enabled via the corresponding configuration block.
  - Validation of the ARC configuration is enforced by A-REX startup scripts. If you have configuration errors then A-REX will not start, and you will be pointed to the error.

- **Scalability and manageability improvements:**
  - The internal job-loop in A-Rex has been re-engineered to be event-driven
  - xrootd plugin has been re-written to improve data transfer performance
  - Consistent handling of server-side logging and improved logfile structure
  - Reworked startup scripts
  - Streamlined package name scheme, where e.g. ldap services have been separated out

- **The RTE framework** has got a redesigned architecture and a largely extended functionality (default, installed and enabled RTEs, introduction of RTE parameters)

- **A new framework for RunTimeEnvironments (RTE):**
  - The RTE framework has gotten a redesigned architecture and largely extended functionality (default, installed, enabled RTEs, introduction of RTE parameters).

**Note:** RTEs must in ARC 6 be explicitly enabled through the new arccitl tool as a separate step after installation.
There are several system RTEs installed together with ARC which you can enable on demand. These are:

- **ENV/CANDYPOND**: makes ARC Candypond (“Cache and deliver your pilot on-demand data”) client available on the Worker Nodes and ready to be used
- **ENV/CONDOR/DOCKER**: enables submission to Docker universe in HTCondor backend
- **ENV/LRMS-SCRATCH**: enables the usage of local WN scratch directory defined by LRMS
- **ENV/PROXY**: copies proxy certificate to the job session directory
- **ENV/RTE**: copies RunTimeEnvironment scripts to the job session directory

**ARCCTL**, a new server-side management and control tool for sysadmins has been developed.

- arcctl is meant to be the central one-stop-shop tool for administrating an ARC CE
- With this tool you can handle RTEs, the jobs, accounting, ARC services and many other things related to an ARC CE.
- arcctl also offers help with integrating 3rd party services and components with an ARC CE such as handling CAs, VOMS, etc..

**JURA Accounting** subsystem improvements:

- Accounting configuration, archive and operations has been restructured to improve typical operations.
- Improved republishing via arcctl.
- Better integration with APEL SSM.
- Archive structure is relying on a database for republishing and gathering stats locally.

**ARCHERY**, the new DNS-based service endpoint catalogue for ARC

- archery-manage package to populate the ARCHERY registry with ARC CE endpoint info
- Official top-level DNS registry under the nordugrid.org domain
- ARCHERY-compatible clients to submit and monitor jobs (arc cli, ldap monitor)
- ARCHERY replaces the OBSOLETEDEGIIS service of the ARC 5 release series.

**TECHNOLOGY PREVIEW** components:

- The REST interface - enabled together with the A-Rex Web-Service.
- The INTERNAL interface - a special interface aimed for restrictive HPC sites, to be used with a local installation of the ARC Control Tower.
- **Candypond** - “Cache and deliver your pilot on-demand data” service to bring the power of ARC Cache to the pilot job universe.
- Experimental Python LRMS for SLURM with SSH access support.

**The ARC client and SDK:**

- The client has undergone a major internal cleanup, large number of submission, target retrieval and job description plugins got removed as a result of OBSOLETING third-party interfaces and technologies in ARC.
- New plugins for ARCHERY and the REST And INTERNAL interfaces were implemented.
- The arccstat cli of the ARC6 client and the underlying SDK now handles walltime as (walltime)*(number of cores) for multicore jobs.
REMOVED and obsoleted components

- Large number of configuration options of arc.conf got DELETED during the server-side configuration rework. A detailed list of those options are available in the arc.conf.DELETED file.
- The separate gangliarc component is discontinued, instead we now offer ganglia integrated into AREX.
- Nordugrid no longer distributes a source code bundle including documentation, gangliarc and nagios.
- CREAM related plugins and server-side components.
- UNICORE related plugins and server-side components.
- All the windows & solaris related ARC code and build.
- JAVA bindings for ARC SDK.
- Support for BES, WSRF and other non-EMIES WS flavours as job management interfaces.
- Support for JDL, ARCJSDL, RSL job description dialects: ARC keeps only XRSI and ADL as supported job description languages.
- EMIR indexing service including ARC CE registration to EMIR and ARC SDK plugins.
- Server-side EGIIS, including ARC CE registration to EGIIS. BUT: keep EGIIS plugins in the client.
- GLUE1 support from the ARC client SDK, server-side support for GLUE1 temporarily kept but labelled as OBSOLETE.
- ARC SDK support to obtain information from top-BDII.
- Publishing nordugrid-authuser objects in the NorduGrid LDAP schema

List of bugs fixed since ARC 5

A rather long (but not exhaustive) list of bugs fixed since ARC 5 can be found here: http://www.nordugrid.org/arc/arc6/common/changelog/bugs-6.0.0.html

7.2.2 List of bugs fixed since ARC 5.4.4

<table>
<thead>
<tr>
<th>Bug ID</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUGZ-895</td>
<td>ARC Administrative tools</td>
</tr>
<tr>
<td>BUGZ-1421</td>
<td>No log of backend script problems</td>
</tr>
<tr>
<td>BUGZ-3034</td>
<td>arcstat -s is case-sensitive</td>
</tr>
<tr>
<td>BUGZ-3106</td>
<td>arcproxy could be more expressive when it finds a problem</td>
</tr>
<tr>
<td>BUGZ-3360</td>
<td>SGE and LL backends fail to report correct node OS/system information in GLUE2 infosystem</td>
</tr>
<tr>
<td>BUGZ-3384</td>
<td>Support for per-queue authorisation configuration and publishing</td>
</tr>
<tr>
<td>BUGZ-3451</td>
<td>Configure number of cpus manually</td>
</tr>
<tr>
<td>BUGZ-3476</td>
<td>Crashes in multiple ARC components</td>
</tr>
<tr>
<td>BUGZ-3545</td>
<td>Patch for Correct Cores Parsing</td>
</tr>
<tr>
<td>BUGZ-3557</td>
<td>undetected job submission in case of heavy filesystem load</td>
</tr>
<tr>
<td>BUGZ-3565</td>
<td>Allow setting default VO in ~/.arc/client.conf to be used in arcproxy</td>
</tr>
<tr>
<td>BUGZ-3566</td>
<td>Implement RTEs processing without shared directory</td>
</tr>
<tr>
<td>BUGZ-3569</td>
<td>Exporting CPU/Wall time limits to Glue2/BDII</td>
</tr>
<tr>
<td>BUGZ-3570</td>
<td>Arcsub used 100GB memory</td>
</tr>
<tr>
<td>BUGZ-3584</td>
<td>JURA: create one log file per job, not per submission</td>
</tr>
<tr>
<td>BUGZ-3624</td>
<td>Data delivery service can only listen to one network interface.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Bug ID</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUGZ-3626</td>
<td>Force GLUE2ComputingManagerTotalLogicalCPUs to be totalcpus when this value is defined in arc.conf</td>
</tr>
<tr>
<td>BUGZ-3632</td>
<td>arcproxy fails in Ubuntu 16.04, 16.10 and recent Debian systems</td>
</tr>
<tr>
<td>BUGZ-3637</td>
<td>arcget with multiple jobs crashes</td>
</tr>
<tr>
<td>BUGZ-3643</td>
<td>The watchdog crashes</td>
</tr>
<tr>
<td>BUGZ-3662</td>
<td>arcsub crashes</td>
</tr>
<tr>
<td>BUGZ-3667</td>
<td>JSON output for arcstat</td>
</tr>
<tr>
<td>BUGZ-3674</td>
<td>settings in client.conf ignored</td>
</tr>
<tr>
<td>BUGZ-3675</td>
<td>Problems retrieving jobs with arcget 1</td>
</tr>
<tr>
<td>BUGZ-3676</td>
<td>Problems retrieving jobs with arcget 2 (first byte missing)</td>
</tr>
<tr>
<td>BUGZ-3677</td>
<td>arcproxy fails retrieving attributes from voms</td>
</tr>
<tr>
<td>BUGZ-3682</td>
<td>Better error message when DN not in gridmap file</td>
</tr>
<tr>
<td>BUGZ-3690</td>
<td>Warnings about missing information on deleted jobs</td>
</tr>
<tr>
<td>BUGZ-3695</td>
<td>Slowness with arccp and the xrootd protocol</td>
</tr>
<tr>
<td>BUGZ-3700</td>
<td>ARC1ClusterInfo.pm uses netstat</td>
</tr>
<tr>
<td>BUGZ-3702</td>
<td>Cannot use arc data commands without certificates</td>
</tr>
<tr>
<td>BUGZ-3707</td>
<td>Seg fault Triolith - related or not to the data-staging problems on Triolith</td>
</tr>
<tr>
<td>BUGZ-3713</td>
<td>Malformed jobs.dat entries</td>
</tr>
<tr>
<td>BUGZ-3722</td>
<td>Xenial repo for the (nordugrid) ARC source is not correct</td>
</tr>
<tr>
<td>BUGZ-3756</td>
<td>package update failed to restart A-REX</td>
</tr>
<tr>
<td>BUGZ-3772</td>
<td>Can’t use different credentials within one submission process when jobs require user input files</td>
</tr>
<tr>
<td>BUGZ-3773</td>
<td>Enabling arex-ganglia breaks controlldir access</td>
</tr>
<tr>
<td>BUGZ-3778</td>
<td>arcctl not working if arc code configured with –disable-swig flag and installed with make install</td>
</tr>
<tr>
<td>BUGZ-3788</td>
<td>Poor performance with arccp and HTTPS</td>
</tr>
<tr>
<td>BUGZ-3812</td>
<td>A-REX hangs inside XRootd after fork</td>
</tr>
</tbody>
</table>

### 7.3 Using ARC packages from nightly builds

Recent development version of ARC is available as a **nightly builds packages** for many Linux distribution.

To use these nightlies smoothly for installation and updates, the repository should be added to your system.

#### 7.3.1 Enabling nightlies repo for RHEL-based distributions

Setup a CRON job (e.g. `/etc/cron.daily/update-arc-nightly-time.sh`) to fetch the latest nightlies date:

```bash
#!/bin/bash
curl -s http://builds.nordugrid.org/nightlies/packages/nordugrid-arc/master/ | sed -n 's/.*<a.*>\(.*\)\</a.*$/\1/p' | sort | tail -1 > /etc/yum/vars/arcnightly
```

Run the CRON script once manually to have the `arcnightly` variable initialized. Also, make sure the script has executable permissions.

To add the nightlies repository to your RHEL-based system, create a `/etc/yum.repos.d/nordugrid-nightly.repo` with the following content:

```
[nordugrid-nightly]
name=Nordugrid ARC Master Nightly Builds - $basearch
baseurl=http://builds.nordugrid.org/nightlies/packages/nordugrid-arc/master/
$arcnightly/centos/el7/$basearch
```

(continues on next page)

---

1 **NOTE** that you should modify OS release version to match your case. For Fedora releases, replace `centos/el` with `fedora/`. 

---

## 7.3. Using ARC packages from nightly builds

---

249
NorduGrid ARC 6 Information, Release ARC6

(continued from previous page)

enabled=1
gpgcheck=0

Check if it works running yum (or dnf), e.g.:
[root ~]# yum makecache fast

7.3.2 Enabling nightlies repo for Debian-based distributions
Nightlies for Debian/Ubuntu are available as standalone packages without repository index files generated.
The suggested approach is to:
• download packages locally
• create the necessary repository package index
• use the repository on the local filesystem
To accomplish this, install the necessary tools for making repository files:
[root ~]# apt-get -y install dpkg-dev

Regularly fetch latest nightly packages and create the repository index with the following daily CRON job:
#!/bin/bash
arcrelease='ubuntu/16.04/amd64'
latestdate=$( wget -q -O - http://builds.nordugrid.org/nightlies/packages/
˓→nordugrid-arc/master/ | sed -n 's/^.*<a.*>\(.*\)\/<\/a>.*$/\1/p' | sort | tail ˓→1 )
repodir=/srv/nordugrid-nightlies
rm -rf $repodir; mkdir -p $repodir; cd $repodir
˓→nordugrid.org/nightlies/packages/nordugrid-arc/master/$latestdate/$arcrelease/
dpkg-scanpackages . /dev/null 2>/dev/null | gzip -9c > Packages.gz

Create the local repository file /etc/apt/sources.list.d/nordugrid-nightlies.list and add
the contents:
deb file:/srv/nordugrid-nightlies ./

Check it works running apt, e.g:
[root ~]# apt-get update

7.4 NorduGrid repository information for ARC 6
The NorduGrid ARC packages are available through YUM and APT repositories for several systems. We have
release-based repositories that you can follow. This will keep your install to a particular release of NorduGrid
ARC with only minor and bug-fixing updates. You can also choose to follow the repository “latest” which will
always point to the latest stable release. For each distribution there are 3 channels (repositories) available:
• base - Base packages (mandatory)
• updates - Updates to the base release (strongly recommended)

250

Chapter 7. Testing Activities


• testing - Packages almost ready to go into the updates repository (optional). The alpha, beta and release candidate releases can be found here.

7.4.1 Repository security

The NorduGrid RPM packages and DEB repositories are signed, and in order for the repository tools APT and YUM to verify them you must install the NorduGrid GPG key:

For rpm based distributions like Red Hat Enterprise Linux and Fedora:

```
[root ~]# rpm --import http://download.nordugrid.org/RPM-GPG-KEY-nordugrid-6
```

For Ubuntu distributions with `sudo`:

```
[user ~]$ wget -q http://download.nordugrid.org/DEB-GPG-KEY-nordugrid-6.asc -O- | sudo apt-key add -
```

For Debian without `sudo`:

```
[root ~]# wget -q http://download.nordugrid.org/DEB-GPG-KEY-nordugrid-6.asc -O- | apt-key add -
```

7.4.2 Repository configuration - Red Hat Enterprise Linux

The NorduGrid ARC repositories for RedHat Enterprise Linux / CentOS packaging utility yum or dnf can be configured through:

```
/etc/yum/nordugrid.repo
```

The repository configuration can be set up automatically by means of installing nordugrid-release package or creating the configuration file manually.

Install nordugrid-release package with YUM/DNF

The easiest way to configure YUM to use the NorduGrid repository for Red Hat Enterprise Linux, CentOS and similar distributions is to install the nordugrid-release package which can be found in the NorduGrid package repository for the appropriate RHEL/EPEL release.

Example packages are shown below for x86_64 architectures, they also exist for i386 when applicable. In that case exchange the x86_64 in the links below with i386.

CentOS: EL7 EL6
Fedora: 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12

Install with `yum` (CentOS) or `dnf` (Fedora), here shown for CentOS:

```
[root ~]# yum install <rhel-repo link>
```

This creates the appropriate repo files in `/etc/yum.repos.d/`

Manual YUM repository setup - NorduGrid repository

For manual YUM repository setup, create a file `/etc/yum.repos.d/nordugrid.repo` with the following contents (here using CentOS as example, if you are on Fedora, replace `centos` with `fedora`)

If you are installing an alpha, beta or release candidate, please set the nordugrid-testing to `enabled=1`.

7.4. NorduGrid repository information for ARC 6
Check if it works running `yum` (or `dnf`), e.g.:

```
[root ~]# yum makecache fast
```

Install required packages

Once the NorduGrid repositories are configured, install the packages with:

```
[root~]# yum install <list of package names>
```

If you are installing an alpha, beta or release candidate, you must install by enabling the nordugrid-testing repo.

```
[root~]# yum install --enablerepo nordugrid-testing <list-of-packages>
```

Are you on recent version of Fedora, use `dnf` instead of `yum`.

Please refer to the *ARC Computing Element Installation and Configuration Guide* for package selection and configuration.

Note that the NorduGrid repositories for RedHat Enterprise Linux/CentOS depends on the EPEL repositories which must also be part of the YUM configuration.

### 7.4.3 Repository configuration - Debian and Ubuntu

The NorduGrid ARC repositories for Debian and Ubuntu packaging utility APT can be configured through:

```
/etc/apt/sources.list
```

or when supported through a repo specific file:

```
/etc/apt/sources.list.d/nordugrid.list
```

The configurations for the varios APT based distributions can be found in the following sections. To enable a specific repository, remove the “#” from the beginning of the line, before the “deb” as shown for the Base Channel.

The repository configuration can be set up automatically by means of installing `nordugrid-release` package or creating the configuration file manually.
Install nordugrid-release package for Debian/Ubuntu through dpkg

The examples below give you the link for most recent Debian/Ubuntu releases. Packages are shown below for amd64 architecture. Replace amd64 for i386 if required for your architecture.

Debian: 9 7 8

Install the source file with dpkg, example shown for Debian 9:

```
[root ~]# wget -q https://download.nordugrid.org/packages/nordugrid-release//releases/6/debian/9/amd64/nordugrid-release_6~bpo9+1_all.deb
[root ~]# dpkg -i nordugrid-release_6~bpo9+1_all.deb
```

For a different version of Debian or Ubuntu, change the version names appropriately.

Manual APT repository setup - NorduGrid repository

For manual APT repository setup for Debian, the APT sources file should contain the following (here shown for Debian 9 stretch):

```
# Base channel - must be enabled
deb http://download.nordugrid.org/repos/6/debian/ stretch main
deb-src http://download.nordugrid.org/repos/6/debian/ stretch main

# Updates to the base release - should be enabled
deb http://download.nordugrid.org/repos/6/debian/ stretch-updates main
deb-src http://download.nordugrid.org/repos/6/debian/ stretch-updates main

# Scheduled package updates - optional
#deb http://download.nordugrid.org/repos/6/debian/ stretch-experimental main
#deb-src http://download.nordugrid.org/repos/6/debian/ stretch-experimental main
```

For manual APT repository setup for Ubuntu, the APT sources file should contain the following (here shown for Ubuntu 18.10 cosmic):

```
# Base channel - must be enabled
deb http://download.nordugrid.org/repos/6/ubuntu/ cosmic main
deb-src http://download.nordugrid.org/repos/6/ubuntu/ cosmic main

# Updates to the base release - should be enabled
deb http://download.nordugrid.org/repos/6/ubuntu/ cosmic-updates main
deb-src http://download.nordugrid.org/repos/6/ubuntu/ cosmic-updates main

# Scheduled package updates - optional
#deb http://download.nordugrid.org/repos/6/ubuntu/ cosmic-experimental main
#deb-src http://download.nordugrid.org/repos/6/ubuntu/ cosmic-experimental main
```

For a different release version, change the version name accordingly.

For Debian:

- 9: stretch
- 8: jessie
- 7: wheezy

For Ubuntu:

- 19.04: disco
- 18.10: bionic
- 18.04: bionic
• 17.10: artful
• 17.04: zesty
• 16.10: yakkety
• 16.04: xenial
• 15.10: wily
• 15.04: vivid
• 14.10: utopic
• 14.04: trusty
• 13.10: saucy
• 13.04: raring
• 12.10: quantal
• 12.04: precise

Install required packages

Run the following command to update the packages database:

```
[root~]# apt-get update
```

Install the packages with (showing example for nordugrid-arc-arex):

```
[root~]# apt-get <list-of-packages>
```

Please refer to the ARC Computing Element Installation and Configuration Guide for package selection and configuration.

7.5 Work-in-progress Docs

Hidden area that holds unfinished documents to be build and available in the doc tree, but not yet linked to the right place.

7.5.1 INTERNAL interface - ARC 6

NB! WIP [TODO] Fix references, now in tex style ref to arc-ce sys admin guide

The INTERNAL submission interface is aimed for restrictive HPC sites. When ARC runs in the internal mode, it should do so alongside a local instance of the ARC Control Tower (aCT). aCT pulls jobs from the central job server (like PaNDA for ATLAS), and and feeds ARC with new jobs internally. Since there is no external access, there is no need for a web-service, gridftp server or ldap, as the purpose of these components are to facilitate external access. There is neither a need for a host-certificate on such a machine. The only service running on ARC is A-REX. Therefore a stripped-down version of aCT and ARC-CE can be used which is beneficial for installation, configuration and maintenance. Furthermore, in the INTERNAL mode ARC should be installed as a normal user. No user mapping (See Section myref{sub:access_control}) is needed in this case, as the default behaviour in ARC 6 is to map the user submitting the job to the A-REX user.
Implementation overview

The INTERNAL submission plugin which is part of the ARC client, interacts with the parent plugin classes using the same API as the other plugins such as gridftp plugin or the emi-es plugin. However, the INTERNAL plugin interacts directly with the A-REX memory and methods, and therefore is integrated as part of the A-REX service which belongs to the ARC-CE code-base. Therefore, both the ARC client and the ARC CE must be installed and on the same machine for the INTERNAL submission plugin interface to function.

All interaction between the client and A-REX happens directly via files in the controldir or via A-REX memory.

Actions

Retrieving Service Information

As a site running in the INTERNAL mode is not accessible from the outside, any service retrieval information can only be done from within the site. You may inspect the service information as usual by issuing the arcinfo command. The information is extracted by direct access to the info.xml file in the contoldir. The sstat method of the INTERNALClient reads the info.xml and outputs information in xml-format to the client. An example of the output of arcinfo for localhost

```
[root ~]# arcinfo -c localhost
Computing service:
  Information endpoint: file://localhost
  Submission endpoint: file://localhost (status: ok, interface: org.nordugrid.internal)
```

When arcinfo is called, the INTERNAL submission interface extracts the site information by directly accessing the info.xml file in the contoldir. The INTERNAL plugin reads the info.xml and outputs information in xml-format to the client, which in turn displays it to the user.

Job submission

When a job is submitted via the INTERNAL submission interface the plugin creates an A-REX job object, which in turn takes care of creating all necessary files (like for instance the ARC job description) and folders (sessiondir) for the job, in addition to creating a job ID. The INTERNAL plugin then places any input files local to the client in the newly created sessiondir. Remaining remote input files are downloaded by the DTR (See Section myref{sub:datastaging}). Once these files are present in the controldir A-REX adds the job to its joblist, and takes over the handling of the job from there.

Accessing Information About Job

Job information evoked by calling arcstat is extracted from a combination of information stored in A-REX memory (job state) and the job.ID.local file in the controldir (session, stagein and stageout directories).

Controlling Execution Of Job

Killing, cleaning and resubmitting jobs is initiated by direct call the existing ARexJob methods: Kill(), Clean(), Resume(). These methods all place files in the controldir that the grid-manager acts upon, such as job.jobid.clean mark or job.jobid.cancel mark.

Delegation Interface [TODO]
BIBLIOGRAPHY