NORDUGRID-TECH-24
3/9/2009

Job Usage Reporter of ARC – JURA

Technical description

Péter Dóbé *

*dobe@iit.bme.hu

1 Introduction

The *Job Usage Reporter of ARC* (JURA) is a component implementing a part of the accounting functionality in the ARC middleware. Its objective is to gather metered resource usage data for each job and submit it to accounting services along with the job submitter's identity and miscellaneous job-related metadata. The collected usage data is transformed into records of job-level granularity and a format corresponding to an OGF specification.

JURA submits the records to an accounting service. The accounting service stores the received usage data in a database, and provides an interface for querying it. Queries can be made by the consumers of the accounting data, such as a billing component. JURA is currently capable of reporting to the logging service of the SweGrid Accounting System (SGAS)[**?**]. Maintaining the possibility to utilise other accounting services has been kept in mind during design, the JURA modular architecture enables easy creation of adapters for accounting services.

JURA offers a functionality similar to that of the *logger*[**?**] client, with improvements and extensions. It also serves as a complete replacement for the JARM component of SGAS.

2 Overview of functionality

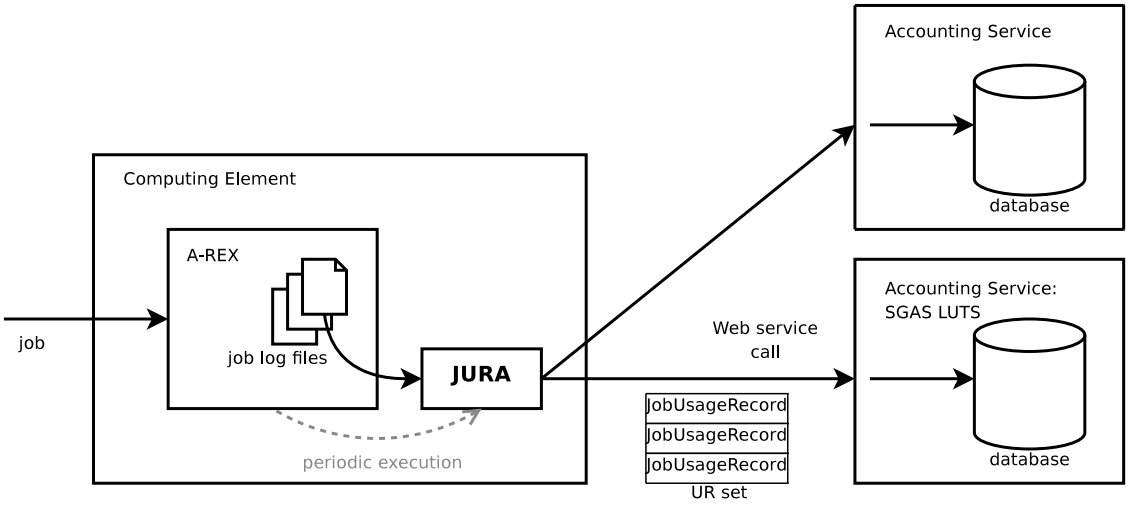

Figure 1: The usage reporting mechanism.

JURA runs as a stand-alone binary application called by A-REX (see Figure 1). There is no designated configuration file for JURA, nor is the configuration file of A-REX read directly by the application. Instead, options related to reporting are included within the job log files generated by A-REX. The primary purpose of these job log files is holding job-related metadata, the main input of JURA. Its format is described in detail in Section 3.2.

The application is run periodically. First, it processes the resource usage content of job log files, and transforms it, generating a standard XML representation of usage data, called Usage Records (URs)[**?**]. Then these records are sent to one or more accounting services, referred to as *reporting destinations* in this document. Several reporting destinations are supported, these can be configured by the system administrator in the A-REX configuration file, and in addition, the user submitting the job can specify destinations in the job description.

Currently, the SGAS Logging and Usage Tracking Service (LUTS) is the only supported reporting destination. Communication with a LUTS server is done via a web service interface. JURA is securely authenticated by the server using X.509 certificates.

3 Operation

3.1 Invocation

JURA is a stand-alone executable application, executed by A-REX hourly (currently a hardcoded time interval, see Section 8). It has no separate configuration file, and does not process the A-REX configuration file. It receives all necessary options from A-REX in part through command-line arguments and mostly via variables inserted into the job log files (See Section 3.2). The following configuration variables can be present in the job log files:

- ***key_path*** – Path to the private key file used when submitting records.
- ***certificate_path*** – Path to the certificate file used when submitting records.
- ***ca_certificates_dir*** – Directory holding the certificates of trusted CAs.
- ***accounting_options*** – Additional configuration options for JURA.

The source of these variables is the "*grid-manager*" block of the A-REX configuration file (see Section 7).

The command line format of JURA is the following:

```
jura [-E <expiration_time>] <control_dir> [<control_dir> [...]]
```

where *expiration_time* is the validity length of job log files in days, after which time they are considered invalid; *control_dir* is the A-REX control directory for a mapped local UNIX user. The "*logs*" subdirectory of each control directory is traversed by JURA separately, in sequence.

3.2 Processing job log files

Job log files contain practically all input data (except those passed as command line arguments) for JURA. A-REX generates these files, at least two for each job and for each reporting destination: one at the time of job submission, another one after the job finishes, and possibly others at each start and stop event. Job log files are the main and only source of detailed resource usage information. Furthermore, they are used to communicate configuration parameters of JURA (see Section 3.1).

The job log files generated by A-REX reside under the directory *<control_dir>/logs*[**?**]. They have file name format *<ngjobid>.<random>*, where *ngjobid* is the identifier created for the job by A-REX, *random* is a randomly generated sequence of alphanumeric characters to avoid collision of different files pertaining to the same job.

A file consists of "*name=value*" lines, where "*value*" is either a job-related resource usage data or a configuration parameter. The URL of the reporting destination corresponding to the job log file is acquired from a "*jobreport=*" line in the A-REX configuration file. In addition to this server-side configuration, a limited number of destinations can be supplied by the submitter in the job description.

JURA generates records in the Usage Record (UR) format[**?**] proposed by the Open Grid Forum (OGF), using the information stored in the job log files. The generated UR is an XML representation holding consumption information for all commonly used resources and metrics. It can be extended by custom elements for non-standard resources and/or other types of job metadata. For a list of UR properties and their sources in the job log file, see Appendix A.

Some elements of UR are mandatory, these must all be present in the job log file to be able to generate a UR. For example, the job log file generated upon job submission contains no *status* entry, so this file is ignored, and no UR is generated from it.

An archiving functionality allows to store generated URs in a specified directory (see Section 7) on the disk. If enabled, valid UR XMLs are written to files named "*usagerecord.<ngjobid>.<random>*", where "*ngjobid*" and "*random*" match those of the source job log file. If a job log file is processed repeatedly – for example because of temporary connection failures to a LUTS service – and a respective UR archive file already exists, then the UR is not generated again. Instead, the contents of the archive file are used without change (NB: the creation time stamp is also retained).

After successful submission to a reporting destination, the job log file is deleted, preventing multiple insertion of usage records. If submission fails, the log files are kept, so another attempt is made upon a subsequent run of JURA. This can repeat until the expiration time passes (see "*-E*" command line switch in Section 3.1), at which point the next execution of JURA removes the file without processing.

3.3 Reporting to LUTS

The generated URs are submitted to the accounting services specified by the reporting destination configuration parameters and if present, to the destinations specified in the job description as well. Reporting URs to several destinations is possible, but currently only SGAS LUTS destinations are supported.

LUTS has a simple custom web service interface loosely based on WS-ResourceProperties[**?**]. JURA uses the insertion method of this interface to report URs. The corresponding job log files are deleted after receiving a non-fault response from the service.

To increase communication efficiency JURA can send URs in batches provided that the server side supports this feature. LUTS 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 generating, but instead collects them in a batch until reaching the maximal number of elements or until running out of job log files. This means that all batches are of maximal size except the last one. The maximal number of URs in a batch can be set as a configuration parameter of JURA ("*jobreport_options*", see Section 7).

4 Security

The JURA executable runs with the same user privileges as the A-REX, typically as *root*. The owner of a job log file is the local user mapped for the submitter entity of the corresponding job. These files contain confidential data, access to which must be restricted, therefore read access is limited to the owner and the super user. If JURA is executed by A-REX, it can read data from these files, and delete expired files.

The authentication towards the SGAS LUTS is done via the standard X.509 certificate mechanism over SSL protocol: a chain of valid (i.e. not expired and/or revoked) certificates with a trusted root certification authority is accepted as authentic identification of the client. In the scenario involving A-REX and JURA, all usage records are submitted using credentials given in the "*jobreport_credentials=*" line of the A-REX configuration file (see A-REX Description and Administration Manual[**?**]), and no proxies are used. Normally the credentials for the A-REX service should be used.

Access control to the LUTS service can only be controlled by very simple means: the Distinguished Name (DN) of the client (in this case JURA) is checked against configured rights. Policies distinguish two rights: publishing and querying. 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. In addition, there is a super user role allowing publishing and querying of any record.

5 Implementation

JURA is written merely in C++, built with widely used GNU tools, and tested in a GNU/Linux environment. It depends on HED libraries: the common utilities library, the message handling library and the Message Chain Components for the TCP, TLS, SOAP and HTTP protocols. It also depends on all mandatory dependencies of ARC: gthread-2.0, glibmm-2.4, libxml-2.0, openssl, e2fsprogs and GNU gettext.

The usage reporting part of the JURA code has a modular design in order to enable adding other types of accounting services besides the currently only supported LUTS. An abstract interface class called "*Destination*" represents a usage reporting destination service. Different inherited classes of "*Destination*" handle different types of services. Currently the only inherited class is "*LutsDestination*", implementing support for LUTS. If one wishes to develop a module for another service type, he or she has to create a new descendant class of "*Destination*", implement its "*report()*" method to submit the contents of the job log file

given as an argument, and adapt the static "*createDestination()*" method of "*Destination*" to instantiate the appropriate class.

6 Installation and deployment

JURA is distributed as part of the ARC technology preview release source tarballs [†]. The component can be built from source tarball using the standard autotools.

The README and INSTALL of the source tarball provide full instructions about building ARC including JURA. The files also list required dependencies (see Section 5).

Upon `make install`, the executable called "*jura*" is placed into the *bin* directory of the configured ARC install location. No other executables or wrapper scripts are installed.

A-REX executes the reporting executable through the file "logger" in the *libexec/arc* directory under the install location (this is an A-REX legacy kept for backwards compatibility with the old logger). Therefore, to enable automatic periodic reporting, a symbolic link called "logger" should be placed in *libexec*, pointing to the JURA executable.

The usage reporting can also be performed manually provided that access to the credentials are granted, by executing JURA with the proper command line arguments (see Section 3.1). The example command below will send generated usage records from the job log files in the standard location, "*/tmp/jobstatus/logs*" and send them to LUTS services. Files older than a week are deleted without processing.

```
jura -E 7 /tmp/jobstatus
```

7 Configuration

JURA can be configured through the configuration file of A-REX[**?**]. As it was already mentioned in Section 3.1, JURA does not process the A-REX configuration file directly; the configuration values are propagated to JURA through the job log files. The following variables in the "*grid-manager*" block of the A-REX configuration file are relevant for JURA:

- ***jobreport****=[URL ... number]* – specifies reporting destination URLs. Multiple entries and multiple URLs are allowed. *number* specifies how long old records have to be kept if failed to be reported. That value is specified in days. Last specified value becomes effective.
- ***jobreport_credentials****=[key_file [cert_file [ca_dir]]]* – specifies the credentials for accessing the accounting service.
- ***jobreport_options****=[options]* – specifies additional options for JURA.

The *jobreport_options* variable allows passing a generic option string to JURA verbatim. This string is interpreted by JURA as a comma-separated list of "*name:value*" pairs (note the colon!), which represent service-related settings and extended reporting parameters. The job reporting options currently defined are:

- ***urbatch****:size* – sets the maximal number of URs in the batch sent within one request. Zero value means unlimited batch size. Default is 50.
- ***archiving****:dir* – enables archiving of generated URs in the given directory. If the directory does not exist, an attempt is made to create it. If this option is absent, no archiving is performed.

The example below is a part of the "*grid-manager*" block of the A-REX configuration. It enables logging of URs to two hosts, using the host credential files (placed in the standard locations), with a maximum of 50 URs per batch. Generated URs are archived in the directory "*/var/urs*". Job log files expire after a week.

[†] http://download.nordugrid.org/software/nordugrid-arc1/trunk/

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

8 Limitations and future plans

Although complete, JURA still has minor imperfections, some of them stemming from limitations or operational characteristics of A-REX or the *arcsub* client:

- The time frequency of running JURA is not configurable. It is a hardcoded value in A-REX: 3600 seconds, i.e. one hour.
- The number of user-supplied reporting destinations is limited for the sake of robustness. This upper limit is hardcoded in A-REX: max. 3 destinations are parsed from JSDL, and max. 1 from RSL.
- The *arcsub* job submission client removes all but one reporting destination URL from the job description, further limiting the number of user-supplied destinations.
- 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.
- Some optional UR properties are not supported (see App. A).
- Memory is not reported correctly. A bug in GNU "time" results in all memory usage set incorrectly as zero.
- Some necessary extensions to the generated UR are not yet filled though the information is already collected in the job log files.
- Detailed user identity information based on the X.509 proxy certificate content or other submitted credentials is missing from URs.

There is also plan to extend JURA with the following features:

- Support for other accounting systems besides LUTS, especially the EGEE Accounting Service. However, appropriate documentation on the service interfaces is missing as of now.
- Extend JURA to be able to read all necessary parameters (credentials, reporting targets) from the command line so that it can serve as a stand-alone reporting utility invoked independently from A-REX.
- Enable generating coarser-grained "*Aggregate URs*" from multiple URs.
- Investigate the subject of project-related charging: who is responsible for determining the "*charge*" value; what rules should be applied?

A Generated Usage Record

The following table shows which properties in OGF UR[**?**] are filled, what data source was used for them, and which properties are missing.

Generated UR Property	**Source (job log entry)**	**Information content**
RecordId	nodename, ngjobid	Globally unique identifier for UR
GlobalJobId	globalid	Globally unique identifier of job: XML element as defined by BES
LocalJobId	localid	CE-specific identifier of job
GlobalUserName	usersn	DN of submitting user's certificate
LocalUserId	localuser	POSIX user on CE executing the job
JobName	jobname	Name of job, as given in job description
Status	status	Status of job
WallDuration [ISO 8601 duration]	usedwalltime [s]	Wall-clock time used by job
CpuDuration [ISO 8601 duration]	usedcputime [s]	CPU time used by job
StartTime [ISO 8601 time stamp]	submissiontime [ISO 8601 time stamp]	Time instant the job started
EndTime [ISO 8601 time stamp]	endtime [ISO 8601 time stamp]	Time instant the job ended
MachineName	nodename	Name of the machine where the job ran (first node from colon-separated list put into element)
Host	nodename	System hostname(s) where the job ran (nodes from colon-separated list put into separate elements)
SubmitHost	clienthost	System hostname the job was submitted from
Queue	lrms	Name of the queue from which the job was executed
ProjectName	projectname	Name of project, as given in job description
Memory [average virtual, kB]	usedmemory [kB]	Average total memory used by job
Memory [max physical, kB]	usedmaxresident [kB]	Maximal resident memory used by job
Memory [average physical, kB]	usedaverageresident [kB]	Average resident memory used by job
NodeCount	nodecount	Number of nodes (physical machines) involved in the job
ProcessID	**MISSING**	The process ID(s) of the job
Charge	**MISSING**	Total charge of the job (money or abstract credits)
Network	**MISSING**	Network usage of job
Disk	**MISSING**	Disk usage of job
Swap	**MISSING**	Swap usage of job
Processors	**MISSING**	Number of processors used or requested
TimeDuration	**MISSING**	Additionally measured time duration(s)
TimeInstant	**MISSING**	Additionally identified time instant(s)
ServiceLevel	**MISSING**	Quality of service associated with usage
Extended UR Property	**Source (job log entry)**	**Description**
RuntimeEnvironment	runtimeenvironment	Requested runtime environment, specified in job description (RTEs from space-separated list put into separate elements)