ARC-XWCH bridge: Running ARC jobs on the XtremWeb-CH volunteer

computing platform

Internal report
Marko Niinimaki, Mohamed BenBelgacem, Nabil Abdennadher
HEPIA, January 2010

1. Background and motivation

The NorduGrid Advanced Resource Connector (ARC) middleware [E2007] is an implementation of fundamental Grid services like information services, resource discovery and monitoring, job submission and management, brokering, data management and resource management. As the name suggests, it is often used to combine the resources of several geographically distributed institutes, to create a grid collaboration. As of late 2009, these collaborations include SWEGRID (Sweden), Swiss National Grid Association (SwiNG), Material Sciences National Grid Infrastructure (M-grid) (Finland), NORGrid (Norway), and the Ukrainian Academic Grid[1]. Currently, ARC can utilize computing resources provided by many cluster computing softwares (so-called Local Resource Management Systems, LRMSs), including Condor, Torque, OpenPBS and Sun Grid Engine [E2007].

The XtremWebCH (XWCH) platform is a cluster computing software inspired by ideas from peer-to-peer and volunteer computing [AB2007]. An XWCH worker node, were computing takes place, can be easily downloaded and run in popular operating systems by individuals and institutes. As a principal platform of the Virtual EZ Grid project[2], it is used in providing hundreds of computing nodes from four Swiss higher education institutes. The use cases for this collaboration come from two directions: (1) porting applications to run "natively" in XWCH and (2) creating a bridge that allows the JOpera [PA2004] users run their jobs in the XWCH platform.

Though hundreds of nodes is a respectable amount, large Grid collaborations typically have thousands. The Swiss Multi-Science Computing Grid (SMSCG) is an ARC collaboration between 11 Swiss higher education institutes connecting (currently) about 3000 CPU's[3]. Enabling SMSCG access to XWCH computing resources is an interesting "pilot project" to combine Grid computing with volunteer computing, and provides added value to both to SMSCG (as resources) and the XWCH developers (as experience in Grid middleware). This has already lead to expanding the capabilities of XWCH, namely by creating a command line interface and job description files (see below).

2. The roles of ARC and XWCH -- the overall view

The aim of the ARC/XWCH integration is that ARC users will submit jobs to ARC servers, as they do now, but that the jobs will be executed on XWCH. A job will be pushed to the XWCH platform by an ARC server (technically: the GridManager of the ARC server). The job is then executed by XWCH, and results retrieved by ARC.

A simple example of an ARC "hello world" job, submitted by a user, is shown below:

```
&("executable" = "/bin/echo" )
("arguments" = "hello, grid" )
("stdout" = "stdout" )
("jobname" = "ARC testjob 2" )
```

There, the first line defines the command that will be run in the node where the job is executed. The second line defines its parameters. The output of the job (stdout) will go to a file called "stdout" (line 3). Line 4 indicates that the job will be reported as "ARC testjob 2" by ARC. For details of the job definition language, see [S4].

The following figure illustrates the role of ARC. It should be noticed that there is nothing XWCH specific in it. There, the user (1) submits a job using the ARC client's "arcsub" command or similar. The client queries the ARC infosystem (2) to find out which ARC servers authorize the user to run jobs, and are otherwise suitable and available to run the job. In (3) a cluster is selected, and the job is executed.

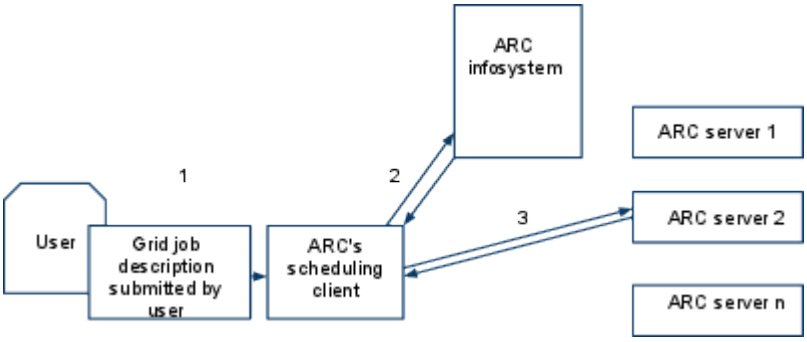

Fig 1: A job in ARC

Each ARC server manages a set of resources, typically a cluster of computers that share a cluster computing software. Figure 2 illustrates what happens to the job in the ARC server. There, the job description file, possibly with local input files arrives in (1) in the server's GridFTP area. GridManager inspects the description file and downloads any external input files that the job needs in the staging area (2). GridManager then adapts the job description to the local resource management system (LRMS), submits the job, and periodically asks the LRMS about the job's status (3). It reports the status to ARC information system (InfoSys), too. When the job is successfully finished, its output files are recovered by GridManager.

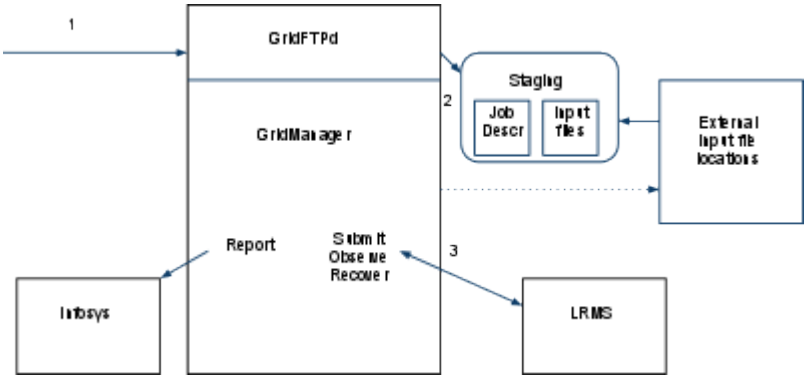

Fig 2: ARC Server and LRMS

Internally at the server, ARC stores a detailed description of the job in a format called GRAMi (GRAM stands for Globus Resource Allocation Manager) [STK2009]. GridManager creates the GRAMi by adapting the incoming job's description in the local system (including an internal ID, the name of the LRMS, and the names of the directories that it

creates for the job). A part of a GRAMi, corresponding to our "hello world" job, can be seen below:

```
joboption_directory='/scratch/grid/271831258642776155450 3709'
joboption_arg_0='/bin/echo'
joboption_arg_1='hello, grid'
joboption_stdin='/dev/null'
joboption_stdout='/scratch/grid/271831258642776155450 3709/stdout'
joboption_stderr='/dev/null'
joboption_count=1
joboption_queue='xwch'
joboption_jobname='ARC testjob 2'
joboption_gridid='271831258642776155450 3709'
```

In the next section we take a detailed view of how job submission, monitoring and recovering the results by GridManager takes place in the case the LRMS is XWCH.

3. Implementation: ARC to XWCH bridge

Technically, GridManager submits, observes and recovers jobs using scripts. These are as follows:
`submit-xxx-job`
`scan-xxx-job`
`finish-xxx-job`
where xxx is the name of the LRMS in question. All these scripts are invoked by GridManager. `submit-xxx-job` is invoked when GridManager receives a job. The other scripts are executed periodically (typically once a minute) and they inform GridManager about status changes, clean up temporary directories and prepare files for upload by the user.

Moreover, for the reporting task, there is a script, simply called `xxxx.pm`, that interfaces the Infosys API [STK2009]. By this script, the LRMS's information about its jobs is passed to ARC InfoSys.

Consequently, scripts `submit-xwch-job`, `scan-xwch-job`, `finish-xwch-job` and `xwch.pm` were created for XWCH. Each of them utilizes the xwch command line interface, that supports the following functionality:

- submitting a job, described in a job description file (see 3.1),
- querying the status of a job (see 3.2),
- getting the results of a job (see 3.3).

3.1 Submitting a GRAMi job to XWCH

When an ARC server receives a job, the GridManager prepares a GRAMi file that is a "localized" job description, in a sense that a temporary directory is allocated to the job, and the input and output files of the job are staged in this directory. The GRAMi file will serve as the starting point of job submission. Specifically, for submitting the job to XWCH, the following steps are taken.

1. A ZIP file is created for the input files of the job.
2. Since XWCH does not support command line arguments, a wrapper script is created for the command line of the job.
3. In XWCH terminology, application, module and job form a hierarchy. Though we only need a simple job, we create application and module for it.

4. The user ID and the address of the XWCH coordinator are read from a configuration file. They are used as parameters for XWCH.
5. The job is submitted to XWCH coordinator. A XWCH job ID is stored in the GRAMi as the internal job ID.

An XWCH internal job description (resulting from steps 2-4 above), and the command to submit it to XWCH is shown below. The command returns an XWCH jobID.

```
type=XWCHJob
os=LINUX
client_ID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx
downloadresults=0
outputfilename=stdout
binaryinputfilename=cline.zip
datainputfilename=data.zip
workercommandline=cline.bat worker
applicationname=testjob
modulename=testjob
jobname=testjob

java -classpath .:....... xwch_cli.Main -f job.xwch
```

3.2 Observing and reporting a job

The `scan-xwch-job` script periodically observes if the job has finished (successfully or unsuccessfully) and if, consequently, its output should be made available to the user. The `xwch.pm` script has another task: it reports to the ARC information system (InfoSys) the status of the XWCH installation in general: how many nodes there are, how many jobs are in the system, and what are their statuses. For both these "scanning" tasks (of scan-xwch-job and xwch.pm), the XWCH command line interface is called by the scripts. The functionality can be summarized as follows:
For each ARC job in the system:

- read the XWCH job ID from the GRAMi file,
- query the status of the XWCH task with that ID,
- tell the result to the InfoSys (by its API).

Technically, one invokes the `xwch_cli.Main` to query about the status of the job:

```
java -classpath .:....... xwch_cli.Main -server <server_address> -status
<jobid>
```

3.3 Recovering job results

When the job has been successfully executed, its results need to be transported to the ARC server so that they can be recovered by the user (who submitted the job). This is done in the following steps:

- Recover the output ZIP file.
- Unzip the file in the ARC job directory.
- Update the GRAMi description so that the status of the job is set to FINISHED.

One invokes the `xwch_cli.Main` as below to download the output:

```
java -classpath .:....... xwch_cli.Main -server <server_address> -d
<jobid;outputfilename>
```

4. Summary and discussion

In this report, we describe the components of ARC-XWCH bridge, a system by which the ARC Grid middleware can submit jobs to the XtremWebCH platform, and recover results from it. This experiment combines ARC, a traditional cluster oriented Grid middleware, with XWCH that is volunteer computing oriented. Because of the different scopes of ARC and XWCH, the following questions will need to be addressed in the future:

- Security: If "anyone" is allowed to join the XWCH platform, and the jobs submitted by ARC process confidential information, how do we protect the information?
- Multi-platform issues: the worker nodes of XWCH are most often normal desktop computers, with "any" operating system system that supports Java. ARC clusters are typically Unix systems. In our current implementation, we limit the XWCH nodes by XWCH's "platform" attribute, so that only worker nodes on Linux are selected. In the future, more platforms should be supported.
- Runtime Environments: ARC relies heavily on Runtime Environments that provide packages for specific applications and programming languages. Currently, such concept is difficult to support with XWCH.

References

[AB2007] N. Abdennadher, R. Boesch, Towards a Peer-To-Peer Platform for High Performance Computing, Grid and Pervasive Computing 07 (GPC'07). May 2007, Paris, France.
[E2007] M. Ellert et al.: Advanced Resource Connector middleware for lightweight computational Grids, Future Generation Computer Systems 23 (2007) 219-240.
[PA2004] C. Pautasso and G. Alonso: JOpera: A Toolkit for Efficient Visual Composition of Web Services, International Journal of Electronic Commerce, vol 9/2, 2004.
[S4] O. Smirnova: XRSL (Extended Resource Specification Language) [NORDUGRID-MANUAL-4]
[STK2009] Ch.U. Søttrup, A. Taga and B. Kónya: ARC Batch System Back-end Interface Guide, Tech Report, NORDUGRID-TECH-13, 2009.

--
1. See http://www.nordugrid.org/about.html
2. See http://www.xtremwebch.net/Projects/Virtual_EZ_Grid
3. See https://cms.smscg.ch and http://elli.switch.ch/