v1.2 October 9, 2014

The ATLAS Connect Virtual Cluster Service

The ATLAS Connect Team

Overview

An ATLAS Connect Virtual Cluster (ACVC) appears as a standard HTCondor pool and is comprised of:

1. A set of resource targets (“remote clusters” or “remote sites”) running various local batch schedulers (HTCondor, SLURM, PBS, LSF, SGE).
2. Corresponding to each is a Remote Cluster Connect Factory (RCCF), a HTCondor service hosted by rccf.usatlas.org which submits glidein jobs (a HTCondor master and startd daemon) to the remote cluster using SSH protocol. A HTCondor process on the compute node reports back to a central collector and becomes part of the user’s HTCondor pool.
3. A login service (login.usatlas.org) from which to submit jobs to the ACVC using standard HTCondor submit scripts.
4. Alternatively, a Tier3 site can be configured to send its jobs to the ACVC via HTCondor flocking. A package of tools is configured centrally is installed on a Tier3 cluster using Git and Puppet (c.f. “thinly provisioned Tier3”). This is called Tier3 Connect and will be described in separate document.

5. A final configuration is one where Panda pilot jobs are submitted to the ACVC using a local Autopyfactory instance, providing a backend job routing service to non-WLCG resources for the CONNECT, CONNECT_CLOUD, ANALY_CONNECT, ANALY_CONNECT_SHORT, and CONNECT_MCORE Panda queues. This enables, for example, a direct connection between Panda and XSEDE resources, as well as university campus clusters.
6. The login service is integrated with the InCommon identity federation via CI-Logon which links to the user's home institution identity service (e.g. Shibboleth). A sign-up service (which does not require a grid certificate) groups users by institutional affiliation or optionally physics working group and auto-provisions Unix accounts for authorized users.
7. All jobs run through the system are tagged and records are kept for accounting purposes using standard tools from HTCondor and OSG.
8. A Ceph object storage service is provided to host quasi-transient job input and output data which has POSIX, http, Xrootd interfaces as well as a Globus Online endpoint for managed transfers to campus data services via ESnet and national facilities such as Blue Waters.
9. Components 2-8 are managed using the MWT2 Git + Puppet configuration system, reside inside LHCONE and are connected to ESnet and Internet2 via an 80 Gbps capacity link to CIC OmniPoP/Starlight Chicago (with a redundant 100 Gbps link to be added in 2015 to 600 W. Chicago, a second OmniPoP switch).

When jobs are submitted to the ACVC from the login service, or flocked into the ACVC from a Tier3 site or Panda, they are routed to resource targets via the RCCF. The RCCF accepts flocked jobs and places them into factories which submit to resource targets based on demand and requirements. The factories describe the resource targets using the standard HTCondor ClassAd mechanism, which includes attributes such as the maximum job runtime, whether jobs are preemptable, or CVMFS availability. There is a small latency introduced as the system warms up: glideins are submitted, remotely scheduled and report back to the factory collector. At that point the normal HTCondor matchmaking process ensues. When there are unclaimed job slots at connected resource targets, users will see their jobs begin running soon after submission, like a normal batch cluster.

Resource Targets

The resource targets are not assumed to be identical and in general there will be sites which don't meet every user's job requirements. Table 1 is a list of factory names available in the ACVC and some of their attributes. All of the factories listed route to sites that have CVMFS available (including ALRB - Atlas Local Root Base) and provide an "Atlas Compatible Environment" (ACE) which is needed at non-WLCG sites. The ACE includes compatibility libraries (HEPOS_libs), grid certificate authority files for X509 authentication (so as to read ATLAS storage via FAX), and the OSG worker node client software. FRONTER_SERVER is an

environment variable pointing to a Squid cache at or nearby the specific resource target. Factories listed in Table 2 do not have CVMFS nor are their OS's ACE compliant.

The "MaxRunTime" is the maximum amount of wall clock time granted to resources routed by this factory. Currently only the Harvard Odyssey resource target will preempt jobs before MaxRunTime has been reached.

If a factory is "MustRequest", the JDL must define a ClassAd specifically asking the RCCF to route the job to the connected resource. This is to exclude as default sites which have preemption, have a very short MaxRunTime, or have other special requirements or requests.

Table 1: Factories (resource targets) with ACE

Factory Name	Resource Target	MaxRunTime	Preemptable	MustRequest
midway	RCC/Midway	30 hours	No	Yes
odyssey	Harvard Odyssey	30 hours	Yes	Yes
icc_taub	Illinois Campus Cluster	3.5 hours	No	Yes
icc_golub	Illinois Campus Cluster	3.5 hours	No	Yes
aglt2	Great Lakes Tier 2	36 hours	No	No
mwt2	Midwest Tier 2	36 hours	No	No
fresnostate	CSU - Fresno State Tier3	36 hours	No	No
utexas	UT Austin Tier3	36 hours	No	Yes

Table 2: Factories (resource targets) without ACE

Factory	Remote Site	MaxRunTime	Preemptable	MustRequest
stampede	TACC/Stampede	24 hours	No	Yes

Job Requirements

One uses HTCondor JDL (Job Description Language) submit files with “Requirements” to be matched against ClassAd definitions to direct jobs to specific resources or resource groups. Each factory has at minimum two ClassAd attributes. They are

- `IS_RCC`
- `IS_RCC_<factory>`

where <factory> is replaced by the factory name associated with a site as listed in Table 1, such as “midway”. These ClassAds are then used in the HTCondor JDL file using a “Requirements” statement

```
Requirements = ( IS_RCC )
```

The above statement will tell HTCondor to flock the job to any factory on the RCCF. To select a specific factory/resource, the statement would be

```
Requirements = ( IS_RCC_midway )
```

With the above statement, the job would only be routed by the “midway” factory, and run on the UChicago midway resource target. To select multiple factories, the HTCondor “OR” statement can be used:

```
Requirements = ( IS_RCC_aglt2 || IS_RCC_mwt2 )
```

With the above statement, the job would only be picked up by the “aglt2” or “mwt2” factory and routed to those Tier2 clusters. The resource which responds first gets the job.

Each factory also has the “HAS_CVMFS” ClassAd defined indicating that CVMFS is available, with the usual ATLAS release directories FUSE-mounted on the resource target, ie:

```
cvmfs2      40G   14G   26G  36% /cvmfs/atlas.cern.ch
cvmfs2      40G   14G   26G  36% /cvmfs/atlas-condb.cern.ch
cvmfs2      40G   14G   26G  36% /cvmfs/sft.cern.ch
cvmfs2      40G   14G   26G  36% /cvmfs/atlas-nightlies.cern.ch
```

Even though all factories have CVMFS available, it is possible that in the future a factory without CVMFS could be added. To force jobs to only run where CVMFS is available, HAS_CVMFS can be used in the Requirement with the HTCondor “AND” statement.

```
Requirement = ( HAS_CVMFS ) && ( IS_RCC )
```

With the above statement, jobs will only be routed by the factories to resource targets which have CVMFS mounted.

ClassAd Definitions

We have decided to mark some factories as MustRequest so they are not included by default but must be specifically requested in the JDL. This is so that the default behavior introduces no surprises for the user. However, more resources can be reached if the user can deal with preemption (odyssey) or short wall clock time limits (icc_taub, icc_golub resource targets on the Illinois Campus Cluster). This also covers the case where the resource owner has requested that jobs only run by request (utexas - the UT Austin Tier3).

For these factories a special ClassAd must be defined by the user in the job JDL file

```
+WANT_<factory> = True
```

For example, for a job to run on the “odyssey” factory, the statement would be

```
+WANT_odyssey = True
```

As many WANT statements can be placed in the JDL, one per line. For example, if the following is in the JDL, the job may run on all factories except “utexas”

```
Requirements    = ( IS_RCC )

+WANT_odyssey   = True

+WANT_midway    = True

+WANT_icc_taub  = True

+WANT_icc_golub = True
```

The “IS_RCC” tells HTCondor to run the job on any available factory. The “+WANT” statements make those restricted factories available for the job. Since “utexas” is not enabled, the job will not run at that restricted factory.

Environment Variables

Before the user job begins execution on the target site, a number of environment variables are predefined. These can be used within the job to determine various characteristics of the factory. This is only a partial list of all defined variables. These are the most relevant to the user.

Table 3: Job Enivironment Variables

Environment Variable	Description
IS_RCC	Job is running via the RCCF (always True)
IS_RCC_<factory>	Job is running in the given factory (always True)
_RCC_Factory	The name of the factory
_RCC_MaxRunningJobs	Maximum number of job which can run in this factory
_RCC_MinSlotLife	Minimum life of a slot to be able to accept a job in minutes
_RCC_FrontierServerURL	List of Frontier Servers
_RCC_FrontierProxyURL	List of Frontier Proxies
_RCC_CVMFS	CVMFS implementation type

Examples

We show here some example uses of JDL and requirements in the ACVC context.

Example 1: hostname.jdl

The following JDL will execute the “hostname” command on the node.

- It will run on any available RCCF factory (Requirements).
- A number of restricted factories will be enabled (+WANT_<factory> = True)
- The log, output and error files will be placed into a “logs” directory (which must exist).
- By using the $(Cluster) and $(Process) expressions, multiple jobs using this JDL will create unique files.
- No notification on job status will be sent via email (Notification = Never)
- Only one job will be submitted per “condor_submit” of this JDL (‘Queue 1)

```
# cat hostname.jdl

Universe                = Vanilla

Requirements            = ( IS_RCC )

+WANT_RCC_icc_taub      = True
```

```
+WANT_RCC_icc_golub     = True
+WANT_RCC_midway        = True
+WANT_RCC_odyssey       = True
+WANT_RCC_utexas        = True
Executable              = /bin/hostname
Should_Transfer_Files   = IF_Needed
When_To_Transfer_Output = ON_Exit
Transfer_Output         = True
Log                     = logs/$(Cluster)_$(Process).log
Output                  = logs/$(Cluster)_$(Process).out
Error                   = logs/$(Cluster)_$(Process).err
Notification            = Never
Queue 1
```

Example 2: verify.jdl

This JDL submits a job to any available factory including many which are restricted.

- The executable is a script called “verify.sh”
- It will also send a file “x509_user_proxy”

```
# cat verify.jdl
Universe                = Vanilla
Requirements            = ( IS_RCC )
+WANT_RCC_icc_taub      = True
+WANT_RCC_icc_golub     = True
+WANT_RCC_midway        = True
+WANT_RCC_odyssey       = True
+WANT_RCC_utexas        = True
Executable              = verify.sh
Should_Transfer_Files   = IF_Needed
```

```
When_To_Transfer_Output = ON_Exit

Transfer_Output         = True

Transfer_Input_Files    = x509_user_proxy


Log                     = logs/$(Cluster)_$(Process).log

Output                  = logs/$(Cluster)_$(Process).out

Error                   = logs/$(Cluster)_$(Process).err

Notification            = Never

Queue 1
```

File: Verify.sh

The following script “verify.sh” will check a health of the CVMFS repositories and then run the Atlas Local Root Base verification tests for Frontier Access.

```
# cat verify.sh

#!/bin/bash

# Verify the health of a CVMFS repository

function f_cvmfs () {

   _cvmfsmount=/cvmfs

   _cvmfsrepo=$1

   _cvmfstdir=$2

   _cvmfstest=${_cvmfsmount}/${_cvmfsrepo}/${_cvmfstdir}

   if [[ ! -e ${_cvmfstest} ]]; then

     echo "+++ERR: CVMFS repository ${_cvmfsrepo} is not available”

   else

     echo "+++INF: CVMFS repository ${_cvmfsrepo} is available"

     echo

     ls -al ${_cvmfsmount}/${_cvmfsrepo}

     echo
```

```
   fi
}
# Link to our proxy
export X509_USER_PROXY=${PWD}/x509_user_proxy

# Setup ARLB
export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
export ALRB_localConfigDir=$PWD/localConfig
source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh

f_cvmfs "atlas.cern.ch"             "repo/sw/software"
f_cvmfs "atlas-condb.cern.ch"       "repo/conditions"
f_cvmfs "atlas-nightlies.cern.ch"   "repo/sw/nightlies"
f_cvmfs "oasis.opensciencegrid.org" "osg"
f_cvmfs "osg.mwt2.org"              "osg"

echo
echo "+++ ALRB diagnostics"
echo

# Get some ALRB diagnostics about this system
localSetupEmi       --quiet

# Check the proxy time
proxyTTL=$(voms-proxy-info --timeleft)

if [[ ${proxyTTL} -lt 60 ]]; then
  echo
  echo "User proxy TTL too short - it dies in ${proxyTTL} seconds"
```

```
  echo
  exit 1
fi
echo
voms-proxy-info -all
echo
diagnostics
echo
echo "Setting up 16.6.7,64,slc5"
echo
asetup 16.6.7,64,slc5
echo
echo "db-readReal"
echo
db-readReal
echo
echo "db-fnget"
echo
db-fnget
echo
```