Submit

Introduction

The submit command provides a means for HUB end users to execute applications on remote resources. The end user is not required to have knowledge of remote job submission mechanics. Jobs can be submitted to traditional queued batch systems including PBS and Condor.

Installation

```
# apt-get install hubzero-app-submit
# apt-get install hubzero-submit-server
# apt-get install hubzero-submit-distributor
```

At completion of the apt-get install commands several files will be located in the directory /opt/submit. Excluding python files the directory listing should like the following:

Configuration

submit provides a mechanism to execute jobs on machines outside the HUB domain. To accomplish this feat some configuration is required on the HUB and some additional software must be installed and configured on hosts in remote domains. Before attempting to configure submit it is necessary to obtain access to the target remote domain(s). The premise is that a single account on the remote domain will serve as an execution launch point for all HUB end users. It is further assumes that access to this account can be made by direct ssh login or using an ssh tunnel (port forwarding).

Having attained account access to one or more remote domains it is possible to proceed with submit configuration. To get started the ssh public generated by the installation should be transferred to the remote domain host(s).

HUB Configuration

The behavior of submit is controlled through a set of configuration files. The configuration files contain descriptions of the various parameters required to connect to a remote domain, exchange files, and execute simulation codes. There are separate files for defining remote sites, staged tools, multiprocessor managers, permissible environment variables, remote job monitors, and ssh tunneling. Most parameters have default values and it is not required that all

parameters be explicitly defined in the configuration files. A simple example is given for each category of configuration file.

Sites

Remote sites are defined in the file sites.dat. Each remote site is defined by a stanza indicating an access mechanism and other account and venue specific information. Defined keywords are

- [name] - site name. Used as command line argument (-v/--venue) and in tools.dat (destinations)
- venues - comma separated list of hostnames. If multiple hostnames are listed one site will chosen at random.
- tunnelDesignator - name of tunnel defined in tunnels.dat.
- siteMonitorDesignator - name of site monitor defined in monitors.dat.
- venueMechanism - possible mechanisms are ssh and local.
- remoteUser - login user at remote site.
- remoteBatchAccount - some batch systems requirement that an account be provided in addition to user information.
- remoteBatchSystem - the possible batch submission systems include CONDOR, PBS, and LSF. SCRIPT may also be specified to specify that a script will be executed directly on the remote host.
- remoteBatchQueue - when remoteBatchSystem is PBS the queue name may be specified.
- remoteBatchPartition - slurm parameter to define partition for remote job
- remoteBatchPartitionSize - slurm parameter to define partition size, currently for BG machines.
- remoteBatchConstraints - slurm parameter to define constraints for remote job
- remoteBinDirectory - define directory where shell scripts related to the site should be kept.
- remoteApplicationRootDirectory - define directory where application executables are located.
- remoteScratchDirectory - define the top level directory where jobs should be executed. Each job will create a subdirectory under remoteScratchDirectory to isolated jobs from each other.
- remotePpn - set the number of processors (cores) per node. The PPN is applied to PBS and LSF job description files. The user may override the value defined here from the command line.
- remoteManager - site specific multi-processor manager. Refers to definition in managers.dat.
- remoteHostAttribute - define host attributes. Attributes are applied to PBS description files.
- stageFiles - A True/False value indicating whether or not files should be staged to remote site. If the the job submission host and remote host share a file system file staging may not be necessary. Default is True.

- passUseEnvironment - A True/False value indicating whether or not the HUB 'use' environment should passed to the remote site. Default is False. True only makes sense if the remote site is within the HUB domain.
- arbitraryExecutableAllowed - A True/False value indicating whether or not execution of arbitrary scripts or binaries are allowed on the remote site. Default is True. If set to False the executable must be staged or emanate from /apps.
- members - a list of site names. Providing a member list gives a layer of abstraction between the user facing name and a remote destination. If multiple members are listed one will be randomly selected for each job.
- state - possible values are enabled or disabled. If not explicitly set the default value is enabled.
- failoverSite - specify a backup site if site is not available. Site availability is determined by site probes.
- checkProbeResult - A True/False value indicating whether or not probe results should determine site availability. Default is True.
- restrictedToUsers - comma separated list of user names. If the list is empty all users may garner site access. User restrictions are applied before group restrictions.
- restrictedToGroups - comma separated list of group names. If the list is empty all groups may garner site access.
- logUserRemotely - maintain log on remote site mapping HUB id, user to remote batch job id. If not explicitly set the default value is False.
- undeclaredSiteSelectionWeight - used when no site is specified to choose between sites where selection weight > 0.

An example stanza is presented for a site that is accessed through ssh.

```
[cluster]
venues = cluster.campus.edu
remotePpn = 8
remoteBatchSystem = PBS
remoteBatchQueue = standby
remoteUser = yourhub
remoteManager = mpich-intel64
venueMechanism = ssh
remoteScratchDirectory = /scratch/yourhub
siteMonitorDesignator = clusterPBS
```

Tools

Staged tools are defined in the file tools.dat. Each staged tool is defined by a stanza indicating an where a tool is staged and any access restrictions. The existence of a staged tool at multiple sites can be expressed with multiple stanzas or multiple destinations within a single stanza. If the tool requires multiprocessors a manager can also be indicated. Defined keywords are

- [name] - tool name. Used as command line argument to execute staged tools. Repeats are permitted to indicate staging at multiple sites.
- destinations - comma separated list of destinations. Destination may exist in sites.dat or be a grid site defined by a ClassAd file.
- executablePath - path to executable at remote site. The path may be given as an absolute path on the remote site or a path relative to remoteApplicationRootDirectory defined in sites.dat.
- restrictedToUsers - comma separated list of user names. If the list is empty all users may garner tool access. User restrictions are applied before group restrictions.
- restrictedToGroups - comma separated list of group names. If the list is empty all groups may garner tool access.
- environment - comma separated list of environment variables in the form e=v.
- softenvExtensions - per site softenv environment declaration for TeraGrid sites running GRAM4.
- remoteManager - tool specific multi-processor manager. Refers to definition in managers.dat. Overrides value set by site definition.
- state - possible values are enabled or disabled. If not explicitly set the default value is enabled.

An example stanza is presented for a staged tool maintained in the yourhub account on a remote site.

```
[earth]
destinations = cluster
executablePath = ${HOME}/apps/planets/bin/earth.x
remoteManager = mpich-intel

[sun]
destinations = cluster
executablePath = ${HOME}/apps/stars/bin/sun.x
remoteManager = mpich-intel
```

Monitors

Remote job monitors are defined in the file monitors.dat. Each remote monitor is defined by a stanza indicating where the monitor is located and to be executed. Defined keywords are

- [name] - monitor name. Used in sites.dat (siteMonitorDesignator)
- venue - hostname upon which to launch monitor daemon. Typically this is a cluster headnode.
- venueMechanism - monitoring job launch process. The default is ssh.
- tunnelDesignator - name of tunnel defined in tunnels.dat.
- remoteUser - login user at remote site.
- remoteMonitorCommand - command to launch monitor daemon process.
- state - possible values are enabled or disabled. If not explicitly set the default value is enabled.

An example stanza is presented for a remote monitor tool used to report status of PBS jobs.

```
[clusterPBS]
venue = cluster.campus.edu
remoteUser = yourhub
remoteMonitorCommand = ${HOME}/SubmitMonitor/monitorPBS.py
```

Multi-processor managers

Multiprocessor managers are defined in the file managers.dat. Each manager is defined by a stanza indicating the set of commands used to execute a multiprocessor simulation run. Defined keywords are

- [name] - manager name. Used in sites.dat and tools.dat.
- computationMode - indicate how to use multiple processors for a single job. Recognized values are mpi, parallel, and matlabmpi. Parallel application request multiprocess have there own mechanism for inter process communication. Matlabmpi is used to enable the an Matlab implementation of MPI.
- preManagerCommands - comma separated list of commands to be executed before the manager command. Typical use of pre manager commands would be to define the environment to include a particular version of MPI amd/or compiler, or setup MPD.
- managerCommand - manager command commonly mpirun. It is possible to include strings that will be sustituted with values defined from the command line.
- postManagerCommands - comma separated list of commands to be executed when the manager command completes. A typical use would be to terminate an MPD setup.
- mpiRankVariable - define environment variable set by manager command to define

process rank. Recognized values are: MPIRUN_RANK, GMPI_ID, RMS_RANK, MXMPI_ID, MSTI_RANK, PMI_RANK, and OMPI_MCA_ns_nds_vpid. If no variable is given an attempt is made to determine process rank from command line arguments.
- environment - comma separated list of environment variables in the form e=v.
- moduleInitialize - initialize module script for sh
- modulesUnload - modules to be unloaded clearing way for replacement modules
- modulesLoad - modules to load to define mpi and other libraries
- state - possible values are enabled or disabled. If not explicitly set the default value is enabled.

An example stanza is presented for a typical MPI instance. The given command should be suitable for /bin/sh execution.

```
[mpich-intel]
preManagerCommands = . ${MODULESHOME}/init/sh, module load mpich-
intel/11.1.038
managerCommand = mpirun -machinefile ${PBS_NODEFILE} -np NPROCESSORS
```

The token NPROCESSORS is replaced by an actual value at runtime.

Environment variables

Legal environment variables are listed in the file environmentwhitelist.dat. The objective is to

prevent end users from setting security sensitive environment variables while allowing application specific variables to be passed to the remote site. Environment variables required to define multiprocessor execution should also be included. The permissible environment variables should be entered as a simple list - one entry per line. An example file allowing use of a variables used by openmp and mpich is presenter here.

```
# environment variables listed here can be specified from the command
line with -e/--env option. Attempts to specify other environment varia
bles will be ignored and the values will not be passed to the remote s
ite.

OMP_NUM_THREADS
MPICH_HOME
```

Tunnels

In some circumstances access to clusters is restricted such that only a select list of machines is allowed to communicate with the cluster job submission node. The machines that are granted such access are sometimes referred to as gateways. In such circumstances ssh tunneling or port forwarding can be used to submit HUB jobs through the gateway machine. Tunnel definition is specified in the file tunnels.dat. Each tunnel is defined by a stanza indicating gateway host and port information. Defined keywords are

- [name] - tunnel name.
- venue - tunnel target host.
- venuePort - tunnel target port.
- gatewayHost - name of the intermediate host.
- gatewayUser - login user on gatewayHost.
- localPortOffset - local port offset used for forwarding. Actual port is localPortMinimum + localPortOffset

An example stanza is presented for a tunnel between the HUB and a remote venue by way of

```
[cluster]
venue = cluster.campus.edu
venuePort = 22
gatewayHost = gateway.campus.edu
gatewayUser = yourhub
localPortOffset = 1
```

Initialization Scripts and Log Files

The submit server and job monitoring server must be started as daemon processes running on the the submit host. If ssh tunneling is going to be used an addition server must be started as a daemon process. Each daemon process writes to a centralized log file facilitating error recording and debugging.

Initialize daemon scripts

Scripts for starting the server daemons are provided and installed in /etc/init.d. The default settings for when to start and terminate the scripts are adequate.

Log files

Submit processes log information to files located in the /var/log/submit directory tree. The exact location varies depending on the vintage of the installation. Each process has its own log file. The three most important log files are submit.log, distributor.log, and monitorJob.log.

submit.log

The submit.log file tracks when the submit server is started and stopped. Each connection from the submit client is logged with the command line and client ip address reported. All log entries are timestamped and reported by client ip address or submit ID once an ID has been assigned. Entries from all jobs are simultaneously reported and intermingled. The submit ID serves as a good search key when tracing problems. Examples of startup, job execution, and termination are given here. The job exit status and time metrics are also recorded in the MyQSL database JobLog table.

```
[Sat Jan 21 14:32:39 2012] Startup: Using configdir /opt/submit
[Sat Jan 21 14:32:39 2012] Startup: Backgrounding process.
[Sat Jan 21 14:32:39 2012] Startup: Listening: protocol='tcp', host=''
```

```
, port=830
[Sat Jan 21 14:32:39 2012] Startup: Listening: protocol='tls', host=''
, port=831




[Thu Feb  2 11:55:57 2012] 128.46.19.176: Connection to tls://:831 fro
m ('128.46.19.176', 49737)
[Thu Feb  2 11:55:57 2012] 128.46.19.176: Server will time out in 60 s
econds.
[Thu Feb  2 11:55:57 2012] 128.46.19.176: Server will time out in 60 s
econds.
[Thu Feb  2 11:55:58 2012] 128.46.19.176: Args are:['/apps/bin/submit'
, '-n', '8', '-v', 'coates', '-w', '4.000000:00:00', '-i', 'BSLAB_512_
RUN', 'bandstrlab-r2091', '/home/nanohub/clarksm/data/sessions/460378L
/BSLAB_512_RUN/OMEN_input_1_512.cmd']
[Thu Feb  2 11:55:58 2012] 2190962: The filesystem is shared.
[Thu Feb  2 12:07:58 2012] 2190962: Job Status: venue=1:sshPBS:3934644
:coates.rcac.purdue.edu status=0 cpu=200.610000 real=36.000000
 wait=619.000000
[Thu Feb  2 12:07:58 2012] 2190962: Server exiting.




[Thu Feb  2 09:38:30 2012] Startup: Server was terminated by a signal.
[Thu Feb  2 09:38:30 2012] Startup: Job Status: venue=any status=65534
 cpu=0.000000 real=0.000000 wait=0.000000
[Thu Feb  2 09:38:30 2012] Startup: EXCEPTION IN MAINLOOP: int argumen
t required
[Thu Feb  2 09:38:30 2012] Startup: Server fell out of mainloop().
[Thu Feb  2 09:38:30 2012] Startup: Server exiting.
```

distributor.log

The distributor.log file tracks each job as it progresses from start to finish. Details of remote site assignment, queue status, exit status, and command execution are all reported. All entries are timestamped and reported by submit ID. The submit ID serves as the key to join data reported in submit.log. An example for submit ID 2190962 is listed here. Again the data for all jobs are intermingled.

```
[Thu Feb  2 11:55:59 2012] 2190962: command = tar vchf 02190962_01_inp
```

```
ut.tar --exclude='*.svn*' -C /home/nanohub/clarksm/data/sessions/46037
8L .__local_jobid.02190962_01 BSLAB_512_RUN -C /home/nanohub/clarksm/d
ata/sessions/460378L/BSLAB_512_RUN OMEN_input_1_512.cmd
[Thu Feb  2 11:55:59 2012] 2190962: remoteCommand bandstrlab-
r2091 ./BSLAB_512_RUN/OMEN_input_1_512.cmd
[Thu Feb  2 11:55:59 2012] 2190962: command = genuserid
[Thu Feb  2 11:55:59 2012] 2190962: IDENTITY = /tmp/id.uBYdxy4FUw
[Thu Feb  2 11:55:59 2012] 2190962: command = update-known-
hosts coates.rcac.purdue.edu
[Thu Feb  2 11:55:59 2012] 2190962: workingDirectory /scratch/lustreA/
n/nano0/nanoHUBjobs/1328219759_02190962_01
[Thu Feb  2 11:55:59 2012] 2190962: command = tar vrhf 02190962_01_inp
ut.tar --exclude='*.svn*' -C /home/nanohub/clarksm/data/sessions/46037
8L/02190962_01 02190962_01.pbs 02190962_01.sh
[Thu Feb  2 11:55:59 2012] 2190962: command = nice -n 19 gzip 02190962
_01_input.tar
[Thu Feb  2 11:55:59 2012] 2190962: command = cat /home/nanohub/clarks
m/data/sessions/460378L/02190962_01/02190962_01_input.tar.gz | ssh -T
-x -a -i /tmp/id.uBYdxy4FUw nano0@coates.rcac.purdue.edu "${HOME}/bin/
receiveinput.sh /scratch/lustreA/n/nano0/nanoHUBjobs/1328219759_021909
62_01 .__timestamp_transferred.02190962_01"
[Thu Feb  2 11:56:01 2012] 2190962: .__local_jobid.02190962_01
[Thu Feb  2 11:56:01 2012] 2190962: command = ssh -T -x -a -i /tmp/id.
uBYdxy4FUw nano0@coates.rcac.purdue.edu "${HOME}/bin/submitbatchjob.sh
 /scratch/lustreA/n/nano0/nanoHUBjobs/1328219759_02190962_01 ./0219096
2_01.pbs"
[Thu Feb  2 11:56:01 2012] 2190962: remoteJobId = 3934644.coates-
adm.rcac.purdue.edu
[Thu Feb  2 11:56:01 2012] 2190962: confirmation: S(1):N Job
[Thu Feb  2 11:56:01 2012] 2190962: status:Job N coates
[Thu Feb  2 11:56:06 2012] 2190962: status:Simulation Q coates
[Thu Feb  2 12:07:42 2012] 2190962: status:Simulation D coates
[Thu Feb  2 12:07:42 2012] 2190962: waitForBatchJobs: nCompleteRemoteJ
obIndexes = 1, nIncompleteJobs = 0, abortGlobal = False
[Thu Feb  2 12:07:42 2012] 2190962: command = ssh -T -x -a -i /tmp/id.
uBYdxy4FUw nano0@coates.rcac.purdue.edu "${HOME}/bin/transmitresults.s
h /scratch/lustreA/n/nano0/nanoHUBjobs/1328219759_02190962_01" | tar x
zmf - --ignore-case --exclude '*hub-
proxy.*' -C /home/nanohub/clarksm/data/sessions/460378L/02190962_01
[Thu Feb  2 12:07:57 2012] 2190962: command = ssh -T -x -a -i /tmp/id.
uBYdxy4FUw nano0@coates.rcac.purdue.edu "${HOME}/bin/cleanupjob.sh /sc
ratch/lustreA/n/nano0/nanoHUBjobs/1328219759_02190962_01"
[Thu Feb  2 12:07:58 2012] 2190962: venue=1:sshPBS:3934644:coates.rcac
.purdue.edu status=0 cputime=200.610000 realtime=36.000000 waittime=61
9.000000 ncpus=8
```

monitorJob.log

The monitorJob.log file tracks the invocation and termination of each remotely executed job monitor. The remote job monitors are started on demand when job are submitted to remote sites. The remote job monitors terminate when all jobs complete at a remote site and no new activity has been initiated for a specified amount of time - typically thirty minutes. A typical report should look like:

```
[Thu Feb  2 11:05:53 2012] (22140) ***********************************
[Thu Feb  2 11:05:53 2012] (22140) * distributor job monitor started *
[Thu Feb  2 11:05:53 2012] (22140) ***********************************
[Thu Feb  2 11:05:53 2012] (22140) loading active jobs
[Thu Feb  2 11:05:53 2012] (22140) 73 jobs loaded from DB file
[Thu Feb  2 11:05:53 2012] (22140) 73 jobs loaded from dump file
[Thu Feb  2 11:05:53 2012] (22140) 2 jobs purged
[Thu Feb  2 11:05:53 2012] (22140) 71 monitored jobs
[Thu Feb  2 11:10:33 2012] (22311) Launching coates
[Thu Feb  2 11:10:33 2012] (22140) 72 monitored jobs
[Thu Feb  2 11:10:44 2012] (22140) Update message received from coates
[Thu Feb  2 11:12:14 2012] (22140) Update message received from coates
[Thu Feb  2 11:18:22 2012] (22629) Launching steele-fe01
[Thu Feb  2 11:18:22 2012] (22140) 73 monitored jobs
[Thu Feb  2 11:19:53 2012] (22140) Update message received from steele-
fe01
[Thu Feb  2 11:21:28 2012] (22140) Update message received from steele-
fe01
[Thu Feb  2 11:50:02 2012] (22629) Closing steele-fe01
[Thu Feb  2 11:51:28 2012] (1420) ***********************************
[Thu Feb  2 11:51:28 2012] (1420) * distributor job monitor stopped *
[Thu Feb  2 11:51:28 2012] (1420) ***********************************
```

It is imperative that the job monitor be running in order for notification of job progress to occur. If users report that their job appears to hang check to make sure the job monitor is running. If necessary take corrective action and restart the daemon.

monitorTunnel.log

The monitorTunnel.log file tracks invocation and termination of each ssh tunnel connection. If users report problems with job submission to sites accessed via an ssh tunnel this log file should be checked for indication of any possible problems.

Remote Domain Configuration

For job submission to remote sites via ssh it is necessary to configure a remote job monitor and a set of scripts to perform file transfer and batch job related functions. A set of scripts can be used for each different batch submission system or in some cases they may be combined with appropriate switching based on command line arguments. A separate job monitor is need for each batch submission system. Communication between the HUB and remote resource via ssh requires inclusion of a public key in the authorized_keys file.

Job monitor daemon

A remote job monitor runs a daemon process and reports batch job status to a central job monitor located on the HUB. The daemon process is started by the central job monitor on demand. The daemon terminates after a configurable amount of inactivity time. The daemon code needs to be installed in the location declared in the monitors.dat file. The daemon requires some initial configuration to declare where it will store log and history files. The daemon does not require any special privileges any runs as a standard user. Typical configuration for the daemon looks like this:

The directory defined by MONITORLOGLOCATION needs to be created before the daemon is started. Sample daemon scripts used for PBS, LSF, Condor, Load Leveler, and Slurm batch systems are included in directory BatchMonitors.

File transfer and batch job scripts

The simple scripts are used to manage file transfer and batch job launching and termination. The location of the scripts is entered in sites.dat.

Examples scripts suitable for use with PBS, LSF, Condor, Load Leveler, and Slurm are included

in directory Scripts. After modifications are made to monitors.dat the central job monitor must be notified. This can be accomplished by stopping and starting the submon daemon or a HUP signal can be sent to the monitorJob.py process.

File transfer - input files

Receive compressed tar file containing input files required for the job on stdin. The file transferredTimestampFile is used to determine what newly created or modified files should be returned to the HUB.

```
receiveinput.sh jobWorkingDirectory transferredTimestampFile
```

Batch job script - submission

Submit batch job using supplied description file. If arguments beyond job working directory and batch description file are supplied an entry is added to the remote site log file. The log file provides a record relating the HUB end user to the remote batch job identifier. The log file should be placed at a location agreed upon by the remote site and HUB.

```
submitbatchjob.sh jobWorkingDirectory jobDescriptionFile
```

The jobId is returned on stdout if job submission is successful. For an unsuccessful job submission the returned jobId should be -1.

File transfer - output files

Return compressed tar file containing job output files on stdout.

```
transmitresults.sh jobWorkingDirectory
```

File transfer - cleanup

Remove job specific directory and any other dangling files

```
cleanupjob.sh jobWorkingDirectory
```

Batch job script - termination

Terminate given remote batch job. Command line arguments specify job identifier and batch system type.

```
killbatchjob.sh jobId jobClass
```

Access Control Mechanisms

By default tools and sites are configured so that access is granted to all HUB members. In some cases it is desired to restrict access to either a tool or site to a subset of the HUB membership. The keywords restrictedToUsers and restrictedToGroups provide a mechanism to apply restrictions accordingly. Each keyword should be followed by a list of comma separated values of userids (logins) or groupids (as declared when creating a new HUB group). If user or group restrictions have been declared upon invocation of submit a comparison is made between the restrictions and userid and group memberships. If both user and group restrictions are declared the user restriction will be applied first, followed by the group restriction.

In addition to applying user and group restrictions another mechanism is provided by the boolean keyword arbitraryExecutableAllowed in the sites configuration file. In cases where the executable program is not pre-staged at the remote sites the executable needs to be transferred along with the user supplied inputs to the remote site. Published tools will have their executable program located in the /apps/tools/revision/bin directory. For this reason submitted programs that reside in /apps are assumed to be validated and approved for execution. The same cannot be said for programs in other directories. The common case where such a situation arises is when a tool developer is building and testing within the HUB workspace environment. To grant a tool developer the permission to submit such arbitrary applications the site configuration must allow arbitrary executables and the tool developer must belong the system group submit.