Submit Command

Overview

submit takes a user command and executes it remotely. The objective is to allow the user to issue a command in the same manner as a locally executed command. Multiple submission mechanisms are available for run dissemination. A set of steps are executed for each run submission:

- Destination site is selected
- A wrapper script is generated for remote execution
- If needed a batch system description file is generated.
- Input files for a run are gathered and transferred to the remote site. Transferred files include the wrapper script and batch description scripts.
- The wrapper script is executed remotely.
- Progress of the remote run is monitored until completion.
- Output files from the run are returned to the dissemination point.

Command Syntax

submit command options can be determined by using the help parameter of the submit command.

```
$ submit --help
Usage: submit [options]

Options:
  -h, --help            Report command usage. Optionally request listi
ng of
                        managers, tools, or venues.
  -l, --local           Execute command locally
  -v, --venue           Remote job destination
  -i, --inputfile       Input file
  -p, --parameters      Parameter sweep variables. See examples.
  -d, --data            Parametric variable data - csv format
  -s SEPARATOR, --separator=SEPARATOR
                        Parameter sweep variable list separator
  -n NCPUS, --nCpus=NCPUS
                        Number of processors for MPI execution
  -N PPN, --ppn=PPN     Number of processors/node for MPI execution
  -w WALLTIME, --wallTime=WALLTIME
                        Estimated walltime hh:mm:ss or minutes
  -e, --env             Variable=value
  -m, --manager         Multiprocessor job manager
  -r NREDUNDANT, --redundancy=NREDUNDANT
```

```
                        Number of indentical simulations to execute in
                        parallel
  -M, --metrics         Report resource usage on exit
  -W, --wait            Wait for reduced job load before submission
  -Q, --quota           Enforce local user quota on remote execution h
ost
  -q, --noquota         Do not enforce local user quota on remote exec
ution
                        host

Parameter examples:

submit -p @@cap=10pf,100pf,1uf sim.exe @:indeck

        Submit 3 jobs. The @:indeck means "use the file indeck as a te
mplate
file." Substitute the values 10pf, 100pf, and 1uf in place of @@cap wi
thin the
file. Send off one job for each of the values and bring back the resul
ts.

submit -p @@vth=0:0.2:5 -p @@cap=10pf,100pf,1uf sim.exe @:indeck

        Submit 78 jobs. The parameter @@vth goes from 0 to 5 in steps
of 0.2,
so there are 26 values for @@vth. For each of those values, the parame
ter
@@cap changes from 10pf to 100pf to 1uf. 26 x 3 = 78 jobs total. Again
@:indeck is treated as a template, and the values are substituted in p
lace of
@@vth and @@cap in that file.

submit -p params sim.exe @:indeck

        In this case, parameter definitions are taken from the file na
med
params instead of the command line. The file might have the following
contents:

                # paramters for my job submission
                parameter @@vth=0:0.2:5
                parameter @@cap = 10pf,100pf,1uf

submit -p "params;@@num=1-10;@@color=blue" job.sh @:job.data

        For someone who loves syntax and complexity... The semicolon s
```

eparates
the parameters value into three parts. The first says to load parameters from
a file params. The next part says add an additional parameter @@num that goes
from 1 to 10. The last part says add an additional parameter @@color with a
single value blue. The parameters @@num and @@color cannot override anything
defined within params; they must be new parameter names.

submit -d input.csv sim.exe @:indeck

Takes parameters from the data file input.csv, which must be in comma-
separated value format. The first line of this file may contain a series of
@@param names for each of the columns. If it doesn't, then the columns are
assumed to be called @@1, @@2, @@3, etc. Each of the remaining lines
represents a set of parameter values for one job; if there are 100 such lines,
there will be 100 jobs. For example, the file input.csv might look like this:

```
@@vth, @@cap
1.1, 1pf
2.2, 1pf
1.1, 10pf
2.2, 10pf
```

Parameters are substituted as before into template files such as
@:indeck.

submit -d input.csv -p "@@doping=1e15-1e17 in 30 log" sim.exe @:infile

Takes parameters from the data file input.csv, but also adds another
parameter @@doping which goes from 1e15 to 1e17 in 30 points on a log scale.
For each of these points, all values in the data file will be executed. If the
data file specifies 50 jobs, then this command would run 30 x 50 = 1500 jobs.

```
submit -d input.csv -i @:extra/data.txt sim.exe @:indeck

        In addition to the template indeck file, send along another fi
le
extra/data.txt with each job, and treat it as a template too.

submit -s / -p @@address=23 Main St.,Hometown,Indiana/42
Broadway,Hometown,Indiana -s , -p @@color=red,green,blue job.sh @:job.
data

        Change the separator to slash when defining the addresses, the
n change
back to comma for the @@color parameter and any remaining arguments. W
e
shouldn't have to change the separator often, but it might come in han
dy if
the value strings themselves have commas.

submit @@num=1:1000 sim.exe input@@num

        Submit jobs 1,2,3,...,1000. Parameter names such as @@num are
recognized not only in template files, but also for arguments on the c
ommand
line. In this case, the numbers 1,2,3,...,1000 are substituted into th
e file
name, so the various jobs take their input from "input1", "input2", ..
.,
"input1000".

submit @@file=glob:indeck* sim.exe @:file

        Look for files matching indeck* and use the list of names as t
he
parameter @@file. Those values could be substituted into other templat
e files,
or used on the command line as in this example. Suppose the directory
contains
files indeckA, indeckB, and indeck-123. This example would launch thre
e jobs
using each of those files as input for the job.
```

Additional information is available by requesting user specific lists of choices for some

command options. The available option lists are generated for a user based on configured restrictions and availability. The values listed here are for example only and may not be available on all HUBs.

```
$ submit --help tools

Currently available TOOLs are:
   pegasus-plan

$ submit --help venues

Currently available VENUES are:
   DiaGrid
   WF-DiaGrid

$ submit --help managers

Currently available MANAGERs are:
   mpi
   mpich
   parallel
```

By specifying a suitable set of command line parameters it is possible to execute commands on configured remote systems. The simple premise is that a typical command line can be prefaced by submit and its arguments to execute the command remotely.

```
$ submit -v clusterA echo Hello world!
Hello world!
```

In this example the echo command is executed on the venue named clusterA where runs are executed directly on the host. Execution of the same command on a cluster using PBS would be done in a similar fashion

```
$ submit -v clusterB echo Hello world!
(2586337) Simulation Queued Wed Oct  7 14:45:21 2009
(2586337) Simulation Done Wed Oct  7 14:54:36 2009
$ cat 00577296.stdout
Hello world!
```

submit supports an extensible variety of submission mechanisms. HUBzero supported submission mechanisms are

- local - use batch submission mechanisms available directly on the submit host. These include PBS, condor, and Pegasus batch queue submission.
- ssh - direct use of ssh. Submit manages access to a common ssh key, essentially serving as a proxy for the HUB user.
- ssh + remote batch submission - use ssh to do batch run submission remotely. Again methods for PBS, condor, and Pegasus batch queue submission are provided.

In addition to single site submission the -r/--redundancy option provides the option to simultaneously submit runs to multiple remote venues. In such cases the successful completion of a run at one venue cancels runs at all other venues. If none of the runs are successful results from one of the runs are returned to the user. Redundant submission is not allowed when performing parametric sweeps.

A site for remote execution is selected in one of the following ways, listed in order of precedence:

- Execute the command within the user tool session, -l/--local option
- User specified on the command line with -v/--venue option.
- Randomly selected from remote sites associated pre-staged application.
- Select randomly from all configured sites

Any files specified by the user plus internally generated scripts are packed into a tarball for delivery to the remote site. Individual files or entire directory trees may be listed as command inputs using the -i/--inputfile option. Additionally command arguments that exist as files or directories will be packed into the tarball. If using ssh based submission mechanisms the tarball is transferred using scp.

The job wrapper script is executed remotely either directly or submitted to a batch queue. The job is subject to all remote queuing restrictions and idiosyncrasies.

Remote batch jobs are monitored for progress. Methods appropriate to the batch queuing system are used to check job status at a configurable frequency. A typical frequency is on the order one minute. Job status changes are reported to the user. The maximum time between reports to the user is set on the order of five minutes even in the absence of change. The job status is used to detect job completion.

The same methods used to transfer input files are applied in reverse to retrieve output files. Any

files and directories created or modified by the application are be retrieved. A tarball is retrieved and expanded to the home base directory. It is up to the user to avoid the overwriting of files.

In addition to the application generated output files additional files are generated in the course of remote run execution. Some of these files are for internal bookkeeping and are consumed by submit, a few files however remain in the home base directory. The remaining files include RUNID.stdout and RUNID.stderr, it is also possible that a second set of standard output/error files will exist containing the output from the batch job submission script. RUNID represents unique job identifier assigned by submit.