Joe Werne, CoRA/NWRA
Chris Bizon CoRA/NWRA
Michael Gourlay CoRA/NWRA

GWD-I
CoRA/NWRA
September 19, 2001

`qprep`: A facility-independent tool for job submission to supercomputers.

September 19, 2001

Status of this memo: Informational

Abstract

U.S. national supercomputer centers present users with a plethora of command syntax, semantics, and functionality for accessing archival data storage. The DoD alone operates and maintains 21 different supercomputer centers for its 5000 users. To simplify job submission for users, the DoD is sponsoring the development of a uniform command-line interface for setting resource limits and job control all of its centers. Here we report the application programming interface (API) `qprep` currently being developed by the DoD so that others may learn of and contribute to the proposed syntax. The intent is to provide a facility-independent interface for the user to control job submission process. We hope that feedback will help shape the API so that it is more broadly used and eventually becomes a standard. We anticipate many grid-related applications share identical needs with those for which `qprep` is currently being developed.

Contents

1. The Problem: Job submission heterogeneity. **1**

2. Proposed Solution: A facility-independent API for job submission. **2**

3. Author Contact Information **3**

4. Acknowledgments **3**

5. Glossary **3**

A. `archive` man page **5**

1. The Problem: Job submission heterogeneity.

Jobs are usually submitted to queues in the form of scripts. These scripts generally fulfill two main goals. First, they define resource limits and job control directives in a comment-style preamble. The specific form of these commands is dependent on the particular queuing system being used at a facility (e.g. NQE, PBS,

LSF). Second, the script itself is run after passing through the system's queue, and must issue commands to run the job, as well as starting any related scripts, running post processing jobs, or performing data archiving.

The tasks associated with resource limits and job controls, though straightforward, are complicated by the differing syntaxes for common tasks offered at different supercomputer centers. For instance, users may have to maintain multiple job control scripts when they work at centers that use different queuing systems such as NQS, PBS, or LSF. When end users who are trained in mathematical and physical sciences (i.e., not computer programming or computer science) confront the script-programming challenges presented by the many differing supercomputer center commands, their solutions are often inefficient and cost them precious labor hours to maintain during the course of their research.

2. Proposed Solution: A facility-independent API for job submission.

Several of the DoD Challenge Users possess supercomputer accounts on multiple platforms at multiple facilities; hence, they are impacted directly by facility heterogeneity. Though a large fraction of such users continue to maintain scripts with hard-coded substitutions to accommodate differences between centers, a subset of Challenge Users have adopted the practice of writing software abstraction layers to convert general user commands into facility-dependent syntax. We have integrated what we feel are the best-practice solutions developed by these users into a portable job submission tool. Users can now write job control scripts in a facility-independent manner by simply using the `qprep` command, which itself handles any facility-dependent behavior.

We are currently in the process of 1) standardizing the software solutions that have arisen naturally from these DoD users, 2) implementing a job submission interface written in Perl (anticpated release in December, 2001), 3) soliciting feedback from users to enhance or modify the API, and 4) implementing the API for NSF or other supercomputer facilities.

`qprep` is a particular application in The Practical Supercomputing Toolkit (www.pstoolkit.org). PST is an open-source project, allowing facilities and users to benefit from the experience of others. Though the DoD has pledged that all of its centers will adopt the API, individual users can download or easily implement solutions for centers that do not yet officially offer the API (because they are not one of the DoD centers and likely they do not yet know the API exists). The choice to allow users to download the software and install it in their home directory was made because users are active in the development, and they do not desire to wait for official sanctioning by a particular center before easily porting code that already uses the `qprep` command interface.

The current man page for the `qprep` API is included as an appendix to this document. `qprep` is a TUSC-level utility of PST that provides an interface between the end-user and a facility's particular preamble syntax. `qprep` reads a script with a facility-independent preamble, and translates it into a script that is directly submittable to a local queue. Alternately, the facility-independent commands may be specified as command line options to `qprep`. `qprep` does not replace the queuing system at a facility, but does allow users to ignore its syntax.

The current list of `qprep` options represent the particular subset of options to NQE and PBS that we have found useful as developers and users at a number of high-performance supercomputing facilities. However, we recognize that our experiences may not cover other common operations, and welcome input at this stage for other options that users desire. Furthermore, one of the options to `qprep` simply passes its

arguments directly to the underlying queue system, in case the local queue system has required functionality not implemented by `qprep`.

With `qprep` and the other TUSC-level applications, users will be able to write job scripts that are completely facility independent. This should allow large productivity increases by first removing the initial learning curve encountered when beginning work on a new center, and second, replacing the maintenance of many facility-dependent scripts with a single facility-independent script. Though the supercomputer application mentioned above has motivated the development of `qprep`, we recognize that job submission presents a set of general problems shared by other applications. We expect that a standard API would help abstract job control components of many other grid-related problems.

3. Author Contact Information

Joe Werne
Colorado Research Associates Division (CoRA),
Northwest Research Associates, Inc. (NWRA),
Boulder, CO 80301
werne@co-ra.com

www.pstoolkit.org contains the man pages for both `archive` and a tool for similarly abstracting the process of job submission (`qprep`), as well as the implementation of `archive` and pointers to mailing lists for PST users and developers.

4. Acknowledgments

Our work on PST is currently funded by the Department of Defense High Performance Computing Modernization Office. In addition to those people listed on this document, significant contributions have been made to PST by Joseph Robichaux, Ray Sheppard, and John Skinner of the Naval Oceanographic Office (NAVO) and David Sanders of the U.S. Army Engineer Research Development Center (ERDC)

5. Glossary

- `archive`: An application in the TUSC layer of PST that provides a facility-independent syntax for dealing with archival data.
- `qprep`: An application in the TUSC layer of PST that translates facility-independent job submission scripts into scripts that will work on a particular facility's machines.
- LSF: A particular queuing system. See www.platform.com for more information.
- NQE: Network Queuing Environment, a particular queuing system.
- PBS: Portable Batch System, a particular queuing system.
- pstrun: An application in the TUSC layer of PST that provides a facility-independent syntax for running a multiprocessor job.

- TUSC: Tools for Uniform Supercomputing. A layer of PST that focuses on hiding facility-dependent details of common tasks such a job submission or data archiving.
- PST: The Practical Supercomputing Toolkit. A collections of utilities that allow users to more productively manage supercomputing resources.

A. `archive` man page

NAME

qprep - Translate job-submission scripts from a platform independent format to the local, platform-dependent format.

SYNOPSIS

qprep [key1=value1[,key2=value2 ...]] *filename*

DESCRIPTION

qprep provides users with a uniform, site-independent command interface to the local queuing system, eliminating the need to learn the myriad syntactical variations that appear at different institutions to perform essentially identical job-submission tasks.

qprep translates a script from a site-independent format (described below) into a site-dependent format, typically one using NQS (Network Queuing System), PBS (Portable Batch System), or a similar queuing system. The translated script is written to a new file, and may optionally be submitted to a queue. Scripts written to the **qprep** standard will be usable at any site where **qprep** is implemented, regardless of the underlying queuing system.

OPTIONS

Options to **qprep** are given in the form of key-value pairs. These may be given either at the command line, seperated by commas without whitespace, or in the preamble to a job script.

Job scripts contain a preamble, similar to those found in native queuing systems. The preamble contains directives that control the resource limits and other attributes of the job. **qprep** directives in the preamble are in the form:

#PSTQ key=value

The # defines the line as a script comment, the string PSTQ alerts **qprep** that the line is a directive that it needs to process, and the actual directive is in the form of a key=value pair, whose possible values are described below.

JOB ENVIRONMENT DIRECTIVES

These directives control the environment in which the script generated by **qprep** operates.

account=account_string

Charge the job to account **account_string**.

username=userlist

Run the job as the specified user. The userlist has the format:

user[@host][,user[@host], ...]

This allows the job to run as different users depending on the host that runs the job. Note, **userlist** is specified with no white space.

exportvars=y

If **exportvars** is set to the string "y", environment variables are exported from the shell in which **qprep** is run to the script's environment.

shell=shell_name

Define the shell that will be used to run the script.

REPORTING DIRECTIVES

These directives control stdout, stderr, and mail reporting for the job.

stderr=*file*

Define the *file* to which stderr is written. This may be the same as the filename for stdout (see below).

stdout=*file*

Define the *file* to which stdout is written. This may be the same as the filename for stderr.

keep=stderr,stdout

Keep either stdout or stderr, or both, on the host where the script was executed. See COMMAND NOTES for a comment on the comma notation used above.

mail=a,b,e,r,t

Send mail at different occurrences. The user may specify multiple triggers separated by commas. See COMMAND NOTES for a discussion of the argument notation.

```
a: send mail when a job is aborted
b: send mail when a job begins execution
e: send mail when a job ends
r: send mail when a job is rerun
t: send mail when a job is routed
```

mailto=user_list

Specifies a comma-separated list (no white space) of users to whom mail is sent. Events that cause mail to be sent are set with mail=key.

JOB CONTROL DIRECTIVES

These items specify particulars of how and where the job is run.

queue=name

Specifies the queue to which the job is submitted.

jobname=name

Specifies a name to associate with the job.

checkpoint=(n|s|number)

Defines the checkpointing characteristics of the job.

```
n: never checkpoint
s: checkpoint when the queue is shut down
number: checkpoint every (number) minutes.
```

NOTE: Not all platforms support checkpointing.

QPREP CONTROL DIRECTIVES

These items specify the behavior of qprep upon translating a script.

silent=y

Do not write the job identifier to stdout when the job is submitted to a queue.

outscript=*outfile*

Write the translated script to *outfile*. If this option is not specified, the translated script is written to *filename*.pst where *filename* is the script argument to **qprep**.

submit=y

After the script has been translated, submit the new script to the queue.

erase=y

Erase the translated script once it has been submitted to the queue. This action is only taken if used in conjunction with submit=y.

LIMIT DIRECTIVES

These items specify resource limits. All times are in the format hours:minutes, where hours and minutes are numbers. All sizes are in MB.

Limits can be specified per-process (a single process within the script) or per-request (the total of all the processes within a script).

nodes=number

Specifies the number of nodes that are simultaneously required by the script. Default is nodes=1.

cpuspernode=number

Specifies the number of cpus per node that are simultaneously required by the script. Default is cpuspernode=1.

walltime=time

Specifies the maximum walltime to be allotted for the script execution.

processcputime=time

Specifies the maximum CPU time used by any single process within the script.

totalcputime=time

Specifies the total CPU time used by all single processes within the script.

procfilesize=size

Specifies the maximum total size of files written by a single process within the script.

totalfilesize=size

Specifies the maximum total size of all files written by all processes within a script.

tapemt(a|b|c|d|e|f|g|h)=number

This set of keys (tapemta, tapemtb, ...) specifies the maximum number of tape drives required in the corresponding device class (a,b, ...). Tape drive device groups can be ascertained using the **tpstat**(1) command.

nice=number

Specifies the nice level at which the script is run when resources are shared. See **nice**(1) for a description of nice levels.

processmemory=size

Specifies the maximum memory used by a single process within the script. **size** is specified in MB. If the process does not share memory with other processes, this option does not need to be specified.

totalmemory=size

Specifies the total memory used by all processes within the script. **size** is specified in MB. If the request does not share memory with other requests, this option does not need to be specified.

JOB DEPENDENCE DIRECTIVES

Sometimes it is necessary for a job to depend in some way upon another job. **qprep** allows jobs to depend upon one another in three ways:

1. 1. Multiple jobs can run at the same time using **synccount** and **syncwith**.
2. 2. Job A can execute only after job B reaches a specified point (dependency is coded in A). This uses the **after** commands (see below).
3. 3. When job B reaches a specified point, it allows job A to run (the dependency is coded in B). This uses the **before** commands (see below).

For **after** and **before**, dependency can be specified on jobs beginning, and ending (with error conditions taken into account). This allows robust scripting in which different jobs run depending on the successful or unsuccessful completion of another job.

The syntax here is very nearly that specified by PBS, which inspired these commands.

synccount=number

Synchronize (**number**) jobs. This is the first job of a set of jobs.

syncwith=jobid

Synchronize with a job in which **synccount** is set.

after=jobid[,jobid, ...]

This job can be executed after all the jobs in the argument have been begun.

afterok=jobid[,jobid, ...]

This job can be executed after all the jobs in the argument have terminated without errors.

afternotok=jobid[,jobid, ...]

This job can be executed after all the jobs in the argument have terminated with errors.

afterany=jobid[,jobid, ...]

This job can be executed after all the jobs in the argument have terminated regardless of errors.

dependson=number

This job can be executed after (**number**) before dependencies (below) have been satisfied.

before=jobid[,jobid, ...]

Jobs **jobid, ...** can be scheduled for execution after the current job begins.

beforeok=jobid[,jobid, ...]

Jobs **jobid, ...** can be scheduled for execution after the current job terminates without errors.

beforenotok=jobid[,jobid, ...]

Jobs **jobid, ...** can be scheduled for execution after the current job terminates with errors.

beforeany=jobid[,jobid, ...]

Jobs **jobid, ...** can be scheduled for execution after the current job terminates regardless of errors.

EXTRA DIRECTIVES

passthrough='arguments'

Pass a quotation-enclosed set of command-line options directly to the underlying queuing system. This is useful if **qprep** does not implement functionality of the underlying system that you require. Also prints a message asking the user to send a request for enhancement to pst-dev@pstoolkit.org.

DIAGNOSTICS

EXAMPLES

ENVIRONMENT

FILES

NOTES

qprep'S ROLE

qprep is a script-editing translation tool in the TUSC layer of the Practical Supercomputing Toolkit (PST). By using **qprep** (and other elements of PST), users develop scripts and techniques that work, without modification, at all institutions supported by PST.

A script written for **qprep** will contain both a preamble and a body. **qprep** translates the preamble only. PST also contains a number of other programs to abstract particular elements of the script body, such as the command to run jobs (e.g. **pstrun**). Currently, the only such command that the core PST team has funding to implement is **pstrun**, but others are possible as future projects.

This page represents an early attempt to standardize the relevant options for **qprep**. If there are options that you will require in order for **qprep** to be useful to you, that are not listed above, please send a request for enhancement to pst-dev@pstoolkit.org describing the problem.

COMMAND NOTES

When a value is shown above as a set of comma-delimeted arguments, one or more of those arguments may be given as the value of the command. For instance, valid specifications of keep would be:

```
keep=stderr
keep=stdout
keep=stdout,stderr
keep=stderr,stdout
```

Possible specifications for mail would include

```
mail=a
mail=a,b
mail=b,e
and so on.
```

SEE ALSO

pstrun: a unified command-line method of running executables.

AUTHOR

Written and Copyright (C) 2001 by Joseph Werne, Michael Gourlay, and Chris Bizon. www.pstoolkit.org.

HISTORY

Documentation v0.1 assembled Friday, July 13, 2001.