BATCH SCHEDULER INTERFACE SPECIFICATION

version 0.3

Michael Frederick

11 September 2005

The first version of the batch scheduler will be command-line only. Once the functionality is there, it should be relatively trivial to replicate the same functionality in a web application or other GUI. I envision that all the functionality will be available through one script, rather than many scripts.

Environment

PLuSH will be utilized to run all jobs and handle setting up the environment, distributing executables, and post-job cleanup. This will greatly simplify the task of the scheduler, which will be used to queue jobs, and fire them off at appropriate times. Most likely Plush will be responsible for setting caps on execution time.

Authentication

Users will only have the ability to view, submit, cancel, and kill their own jobs. We want to use PLuSH's authentication methods to keep things simple. The best way to do this is to run PLuSH as the user submitting the job. This means that we need to somehow store the private key or passphrase for later use.

Priority

In the first version of the scheduler, users' jobs will have a default priority. The scheduler will process jobs in a single round-robin queue based on the time that the job was input. There will be some facility for backfilling to maintain queue efficiency.

Control Flow

The scheduler will run as a daemon on the head controller for the cluster, where the PLuSH controller resides. The scheduler will communicate with the controller via the command-line functionality of PLuSH. Scheduler clients will communicate with the scheduler through the command-line scripts as outlined below. The server will operate an XML-RPC server over HTTP, which the client scripts can connect with to pass information back and forth. This will allow the clients to run on any remote station with a Python installation.

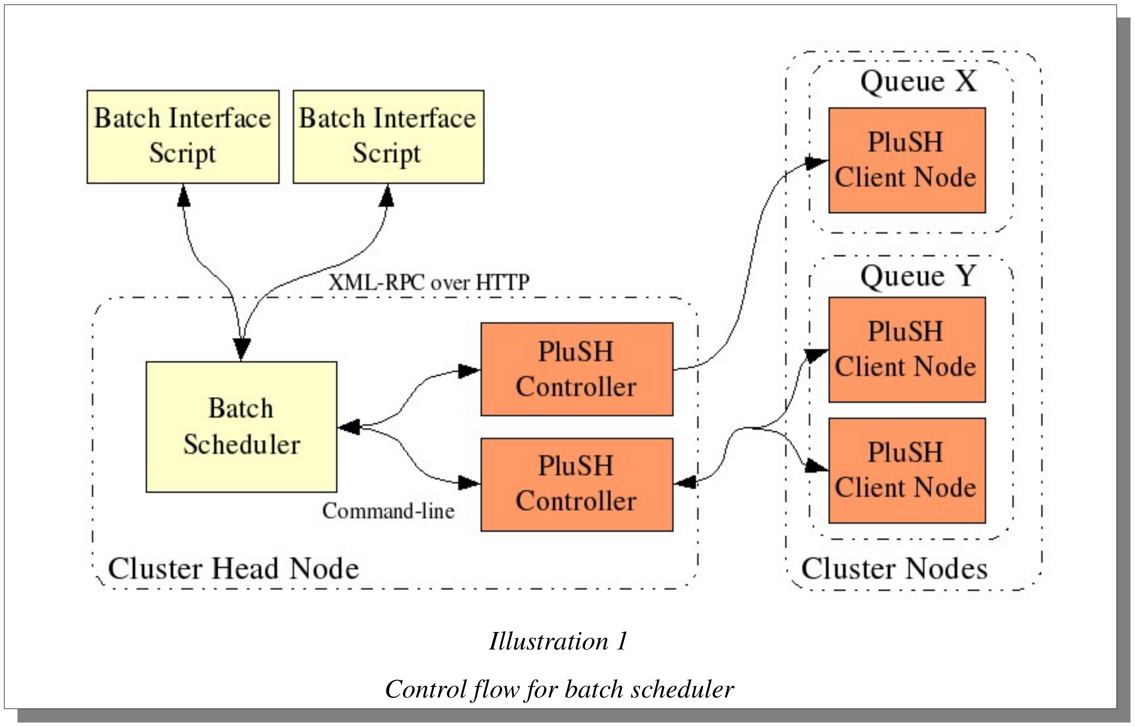

Illustration 1

Control flow for batch scheduler

Scheduler to PLuSH communication

Command	***PLuSH command-line interface***
starting a job	● `./plush` ● `load <plush file>` ● `select project "demo"` ● `select experiment "demo_expt"` ● `run`
killing a job	● `shell 'killall <process name>'` ● `disconnect`
status of a running job	● `shell 'ps -ef | grep <process name>'`

Job storage

The scheduler will keep a database of the pending and completed jobs. This will be stored in an XML file on the head node, and will only be modified through the scheduler interface. After a job submission or cancellation, the database will be changed to reflect the change in job status.

Server usage:

```
server.py [options]

options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -p PORT, --port=PORT  port number to listen on
  -f FILE, --file=FILE  local database xml
```

Port and file options are required.

Client usage:

Here are the major tasks and possible command line parameters associated with each:

Help and usage

```
scheduler help
```

Creating a job

To submit a new PluSH job:

```
scheduler submit -n <name of job> -f <plush xml file> [-e <email when
done>]
```

Viewing job info

Generic syntax:

```
scheduler status ( -a | -n <name of job> | -i <job id> | -u <userid> |
-m <machine name> ) [-e <email status>]
```

Show the status of all jobs in the queue which the user has permission to see:

```
scheduler status -a
```

Show the status of a specific job in the queue based on job name:

```
scheduler status -n <name of job>
```

Show the status of a specific job in the queue based on job id:

```
scheduler status -i <id of job>
```

Show the status of all jobs belonging to a specific user:

```
scheduler status -u <userid>
```

Show the status of a machine in the cluster:

```
scheduler status -m <machine name>
```

Killing a job

Generic syntax:

```
scheduler kill ( -a | -n <name of job> | -i <job id> | -u <userid> | -m
<machine name> ) [-e <email status>]
```

Kill a job based on the job's name:

```
scheduler kill -n <name of job>
```

Kill a job based on the job's numerical id:

```
scheduler kill -i <id of job>
```

Kill all jobs under user's control:

```
scheduler kill -a
```

Cancelling jobs

Generic syntax:

```
scheduler cancel ( -a | -n <name of job> | -i <job id> | -u <userid> |
-m <machine name> ) [-e <email status>]
```

Cancel an upcoming job based on the job's name:

```
scheduler cancel -n <name of job>
```

Cancel an upcoming job based on the job's numerical id:

```
scheduler cancel -i <job id>
```

Cancel all upcoming jobs that the user has access to:

```
scheduler cancel -a
```

```
sched.py command [options]

commands:
  cancel
  submit
  kill
  status

options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -e EMAIL, --email=EMAIL
                        email list for command results
  -f FILE, --file=FILE  PLuSH XML file for experiment
  -i JID, --id=JID      job id to operate on
  -n JNAME, --name=JNAME
                        job name to operate on
  -u USER, --user=USER  username to operate on
  -a, --all             specifies all
  -p PASSPHRASE, --passphrase=PASSPHRASE
                        passphrase for job submission
```

Scheduler commands

Command	***Description***
`cancel`	cancels a pending job
`help`	display usage help
`kill`	kills a currently running job
`status`	shows status, based on parameters
`submit`	submit a new job

Example walkthrough

Scenario: User has group of machines defined as a particular queue. User wants to run an experiment on this queue using the scheduler to run at an appropriate time.

The user creates an experiment file through the PLuSH interface called plush1.xml, and wants it to run as soon as possible.

```
host$ scheduler submit -n “test run 1” -f plush1.xml -e
mjfreder@cs.ucsd.edu
Submitted: job “test run 1” with JID 12345.
```

Now the scheduler has added the job, and will run it at the next opportunity. mjfreder@cs.ucsd.edu will receive a confirmation email when the job completes.

Oops! The user just realized that he wanted his advisor to see the output of the job as well. The user also forgot the name of the job he just created, but remembers the PID. First, the user cancels the job request:

```
scheduler cancel -i 12345
Cancelled: job “test run 1” with JID 12345.
```

... then re-adds the job:

```
host$ scheduler submit -n “test run 1” -f plush1.xml -e
mjfreder@cs.ucsd.edu,vahdat@cs.ucsd.edu
Submitted: job “test run 1” with JID 12346.
```

The user now wants to check on the status of the job:

```
host$ scheduler status -n “test run 1”
Pending: job “test run 1” with JID 12346
```

The user checks again, a while later:

```
host$ scheduler status -n “test run 1”
Running: job “test run 1” with JID 12346
```

The user now realizes that the experiment is flawed after going through some logs. He decides to kill the experiment:

```
host$ scheduler kill -n “test run 1”
Killed: job “test run 1” with JID 12346
```