Running Jobs with Platform Lava

Version 1.0
September 2007
Comments to: doc@platform.com

Platform

We'd like to hear from you

You can help us make this manual better by telling us what you think of the content, organization, and usefulness of the information. If you find an error, or just want to make a suggestion for improving this manual, please address your comments to doc@platform.com.

Your comments should pertain only to Platform documentation. For product support, contact support@platform.com.

Although the information in this document has been carefully reviewed, Platform Computing Inc. ("Platform") does not warrant it to be free of errors or omissions. Platform reserves the right to make corrections, updates, revisions or changes to the information in this document.

UNLESS OTHERWISE EXPRESSLY STATED BY PLATFORM, THE PROGRAM DESCRIBED IN THIS DOCUMENT IS PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL PLATFORM COMPUTING BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING WITHOUT LIMITATION ANY LOST PROFITS, DATA, OR SAVINGS, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS PROGRAM.

Last update

July 31 2007

Contents

CHAPTER 4

Running Jobs

Contents

Submitting Jobs

Submitting a job

You submit a job with the `bsub` command. If you do not specify any options, the job is submitted to the default queue configured by the Lava administrator (usually the `normal` queue).

For example, if you submit the job `my_job` without specifying a queue, the job goes to the default queue.

```
$ bsub my_job
Job <1234> is submitted to default queue <normal>
```

In the above example, 1234 is the job ID assigned to this job, and `normal` is the name of the default job queue.

See the `bsub` command in the Platform Lava man pages for more details on `bsub` options.

Submitting a script

Any command or script you can execute from a shell prompt can be submitted to Lava for batch execution.

To submit a script to Lava:

1. Create a script. For example, create the following script and save it as `myscript`.

   ```
   #!bin/sh
   #BSUB -q test
   #BSUB -o outfile -R "mem>10"
   myjob arg1 arg2
   #BSUB -J myjob
   ^D
   ```

2. Make the script executable. For example:

   ```
   $ chmod u+x myscript
   ```

3. Submit the script to Lava:

   ```
   $ bsub < myscript
   Job <1234> is submitted to queue <normal>.
   ```

Note

- Command-line options override embedded options.
- Submission options can be specified anywhere in the standard input. In the above example, the `-J` option of `bsub` is specified after the command to be run.
- More than one option can be specified on one line.

Submitting interactive jobs

Use the `bsub -I` option to submit batch interactive jobs.

For more details, see the `bsub(1)` man page.

Interactive batch jobs cannot be rerunnable (`bsub -r`) or submitted to rerunnable queues (RERUNNABLE=y in `lsb.queues`).

Submitting a job to run under a particular shell

By default, Lava runs batch jobs using the Bourne (`/bin/sh`) shell. You can specify the shell under which a job is to run. This is done by specifying an interpreter in the first line of the script.

For example:

```
$ bsub
bsub> #!/bin/csh -f
bsub> set coredump=`ls |grep core`
bsub> if ( "$coredump" != "") then
bsub> mv core core.`date | cut -d" " -f1`
bsub> endif
bsub> myjob
bsub> ^D
Job <1234> is submitted to default queue <normal>.
```

The `bsub` command must read the job script from standard input to set the execution shell. If you do not specify a shell in the script, the script is run using `/bin/sh`. If the first line of the script starts with a `#` not immediately followed by an exclamation mark (`!`), then `/bin/csh` is used to run the job.

For example:

```
$ bsub
bsub> # This is a comment line. This tells the system to use /bin/csh to
bsub> # interpret the script.
bsub>
bsub> setenv DAY `date | cut -d" " -f1`
bsub> myjob
bsub> ^D
Job <1234> is submitted to default queue <normal>.
```

If you frequently need to run jobs under a particular shell, you can specify an alternate shell using a command-level job starter and run your jobs interactively. For information on command-level job starters, see “Job Starters” on page 16.

Submitting a job to specific hosts

Submitting a job to a single host

To indicate that a job must run on one of the specified hosts, use the `bsub -m "hostA hostB ..."` option.

By specifying a single host, you can force your job to wait until that host is available and then run on that host.

For example:

```
$ bsub -q idle -m "hostA hostD hostB" myjob
```

This command submits `myjob` to the idle queue and tells Lava to choose one host from `hostA`, `hostD`, and `hostB` to run the job. All other batch scheduling conditions still apply, so the selected host must be eligible to run the job.

Tip If you have applications that require specific resources, create a new Boolean resource. For more information, see *Inside Platform Lava*.

Submitting a job with resource requirements

To submit a job that will run on 32-bit Linux or 64-bit Linux:

```
$ bsub -R "type==LINUX86 || type==LINUX64" myjob
```

When you submit a job, you can also exclude a host by specifying a resource requirement using `hname` resource:

```
$ bsub -R "hname!=hostb && type==LINUX86" myjob
```

See below for more information on submitting jobs with resource requirements.

Specifying resource requirements for a job

Each job can specify resource requirements. Resource requirements specified for a job override any resource requirements specified in the remote task list.

In some cases, the queue specification sets an upper or lower bound on a resource. If you attempt to exceed that bound, your job will be rejected.

Syntax

To specify resource requirements for your job, use `bsub -R` and specify the resource requirement string.

Examples

```
$ bsub -R "swp > 15 && linux order[cpu]" myjob
```

This runs `myjob` on a Linux host that is lightly loaded (CPU utilization) and has at least 15 MB of swap memory available.

Also see “Submitting a job to specific hosts” on page 7 for the two examples using resource requirements.

Specifying resource usage limits for a job

To specify resource usage limits at the job level, use one of the following `bsub` options:

- `-C` *core_limit*
- `-c` *cpu_limit*
- `-D` *data_limit*
- `-F` *file_limit*
- `-M` *mem_limit*
- `-W` *run_limit*
- `-S` *stack_limit*
- `-v` *swap_limit*

Job-level resource usage limits specified at job submission override the queue definitions.

For supported resource usage limits and syntax, see the Platform Lava man pages.

Running Parallel Jobs

Specifying the number of processors

When submitting a parallel job that requires multiple processors, you can specify the exact number of processors to use.

To submit a parallel job, use `bsub -n` and specify multiple processors.

Example

```
$ bsub -n 4 myjob
```

This command submits myjob as a parallel job. The job is started when 4 job slots are available.

Job slot limits for parallel jobs

A job slot is the basic unit of processor allocation in Lava. A sequential job uses one job slot. A parallel job that has *N* components (tasks) uses *N* job slots, which can span multiple hosts.

Job submission examples

Submitting a job to a specific queue

If you have an urgent job `my_job` to run, you can submit it to the priority queue:

```
$ bsub -q priority my_job
```

If you want to use hosts owned by others and you do not want to bother the owners, you can run your low priority jobs on the idle queue so that as soon as the owner comes back, your jobs get suspended.

Submitting a job with a start time

If you do not want to start your job immediately when you submit it, use `bsub -b` to specify a start time. Lava will not dispatch the job before this time. For example:

```
$ bsub -b 5:00 myjob
```

This example submits a job that remains pending until after the local time on the master host reaches 5 a.m.

Submitting a job with an end time

Use `bsub -t` to submit a job and specify a time after which the job should be terminated. For example:

```
$ bsub -b 11:12:5:40 -t 11:12:20:30 myjob
```

The job called `myjob` is submitted to the default queue and will start after November 12 at 05:40 a.m. If the job is still running on November 12 at 8:30 p.m., it will be killed.

Submitting a batch interactive job

```
$ bsub -I -q interactive -n 4,10 myapp
```

This example starts myapp on 4 to 10 processors and displays the output on the terminal.

Modifying jobs

If your submitted jobs are pending (`bjobs` shows the job in PEND state), use the `bmod` command to modify job submission parameters.

See the `bmod` command in the Platform Lava man pages for more details.

Changing a job parameter

To change a specific job parameter, use `bmod` with the `bsub` option used to specify the parameter. The specified options replace the submitted options. The following example uses the `-b` option to change the start time of job 101 to 2:00 a.m.:

```
$ bmod -b 2:00 101
```

Resetting

To reset an option to its default submitted value (undo a `bmod`), append the `n` character to the option name, and do not include an option value. The following example resets the start time for job 101 back to its default value:

```
$ bmod -bn 101
```

Job Dependency Scheduling

Sometimes, the scheduling of a job depends on the result of another job. For example, a series of jobs could process input data, run a simulation, generate images based on the simulation output, and finally, record the images on a high-resolution film output device. Each step can only be performed after the previous step finishes successfully, and all subsequent steps must be aborted if any step fails.

Some jobs may not be considered complete until some post-job processing is performed. For example, a job may need to exit from a post-execution job script, clean up job files, or transfer job output after the job completes.

In Lava, any job can be dependent on other Lava jobs. Lava will not place your job unless this dependency expression evaluates to TRUE. If you specify a dependency on a job that Lava cannot find (such as a job that has not yet been submitted), your job submission fails.

Specifying a job dependency

To specify job dependencies, use `bsub -w`.

Syntax **`bsub -w '`***`dependency_expression`***`'`**

The dependency expression is a logical expression composed of one or more dependency conditions. For syntax of individual dependency conditions, see “Dependency conditions” on page 12.

To make a dependency expression of multiple conditions, use the following logical operators:

- && (AND)
- || (OR)
- ! (NOT)

- If necessary, use parentheses to indicate the order of operations.
- Enclose the dependency expression in single quotes (') to prevent the shell from interpreting special characters (space, any logic operator, or parentheses). If you use single quotes for the dependency expression, use double quotes for quoted items within it, such as job names.
- Job names specify only your own jobs unless you are a Lava administrator.
- Use double quotes (") around job names that begin with a number.
- In the job name, specify the wildcard character (*) at the end of a string to indicate all jobs whose name begins with the string. For example, if you use `jobA*` as the job name, it specifies jobs named `jobA`, `jobA1`, `jobA_test`, and `jobA.log`.

Dependency conditions

The following dependency conditions can be used with any job:

- **done(***job_ID* | **"***job_name***")**
- **ended(***job_ID* | **"***job_name***")**
- **exit(***job_ID* [**,**[*op*] *exit_code*]**)**
- **exit("***job_name***"**[**,**[*op*] *exit_code*]**)**
- *job_ID* | **"***job_name***"**
- **post_done(***job_ID* **|** **"***job_name***")**
- **post_err(***job_ID* **|** **"***job_name***")**
- **started(***job_ID* **|** **"***job_name***")**

done

Syntax **done(***job_ID* | **"***job_name***")**

Description The job state is DONE.

ended

Syntax **ended(***job_ID* | **"***job_name***")**

Description The job state is EXIT or DONE.

exit

Syntax **exit(***job_ID* | **"***job_name***"**[**,**[*operator*] *exit_code*]**)**

where *operator* represents one of the following relational operators:

- >
- >=
- <
- <=
- ==
- !=

Description The job state is EXIT, and the job's exit code satisfies the comparison test.

If you specify an exit code with no operator, the test is for equality (== is assumed).

If you specify only the job, any exit code satisfies the test.

Examples

- `exit (myjob)`

  The job named `myjob` is in the EXIT state, and it does not matter what its exit code was.

- `exit (678,0)`

  The job with job ID 678 is in the EXIT state, and terminated with exit code 0.

- `exit ("678",!=0)`

The job named `678` is in the EXIT state, and terminated with any non-zero exit code.

Job ID or job name

Syntax *job_ID* | **"***job_name***"**

Description If you specify a job without a dependency condition, the test is for the DONE state (Lava assumes the “done” dependency condition by default).

post_done

Syntax **post_done(***job_ID* **| "***job_name***")**

Description The job state is POST_DONE (the post-processing of specified job has completed without errors).

post_err

Syntax **post_err(***job_ID* **| "***job_name***")**

Description The job state is POST_ERR (the post-processing of specified job has completed with errors).

started

Syntax **started(***job_ID* **| "***job_name***")**

Description The job state is:

- RUN, DONE, or EXIT
- PEND or PSUSP, and the job has a pre-execution command (`bsub -E`) that is running

Pre-Execution and Post-Execution Commands

Each batch job can be submitted with optional pre- and post-execution commands. Pre-execution and post-execution commands can be any executable command lines to be run before a job is started or after a job finishes.

Some batch jobs require resources that Lava does not directly support. For example, appropriate pre-execution and/or post-execution commands can be used to handle various situations:

- Reserving devices like tape drives
- Creating and deleting scratch directories for a job
- Customizing scheduling
- Checking availability of software licenses
- Assigning jobs to run on specific processors on SMP machines

By default, the pre- and post-execution commands are run under the same user ID, environment, and home and working directories as the batch job is run. If the command is not in your normal execution path, the full path name of the command must be specified.

To configure pre-execution and post-execution commands, see *Inside Platform Lava*.

Pre-execution commands

The pre-execution command returns information to Lava using its exit status. When a pre-execution command is specified, the job is held in the queue until the specified pre-execution command returns exit status zero (0).

If the pre-execution command exits with non-zero status, the batch job is not dispatched. The job goes back to the PEND state, and Lava tries to dispatch another job to that host.

If the pre-execution command exits with a value of 99, the job will not go back to the PEND state; it will exit. This gives you flexibility to abort the job if the pre-execution command fails.

Post-execution commands

If a post-execution command is specified, then the command is run after the job is finished regardless of the exit state of the job.

Post-execution commands are typically used to clean up some state left by the pre-execution and the job execution. Post-execution is only supported for a queue—not for a specific job. For queue-level commands, see *Inside Platform Lava*.

Submitting a job with a pre-execution command

The `bsub -E` option specifies an arbitrary command to run before starting the batch job. When Lava finds a suitable host on which to run a job, the pre-execution command is executed on that host. If the pre-execution command runs successfully, the batch job is started.

Job-level post-execution commands are not supported.

Post-execution job states

Some jobs may not be considered complete until some post-job processing is performed. For example, a job may need to exit from a post-execution job script, clean up job files, or transfer job output after the job completes.

The DONE or EXIT job states do not indicate whether post-processing is complete, so jobs that depend on processing may start prematurely. Use the `post_done` and `post_err` keywords on the `bsub -w` command to specify job dependency conditions for job post-processing. The corresponding job states POST_DONE and POST_ERR indicate the state of the post-processing. See "Dependency conditions" on page 12 in the section on job dependency scheduling.

The `bhist` command displays the POST_DONE and POST_ERR states. The resource usage of post-processing is not included in the job resource usage.

After the job completes, you cannot perform any job control on the post-processing. Post-processing exit codes are not reported to Lava. The post-processing of a repetitive job cannot be longer than the repetition period.

Job Starters

Some jobs have to run in a particular environment, or require some type of setup to be performed before they run. In a shell environment, job setup is often written into a wrapper shell script file that itself contains a call to start the desired job.

A *job starter* is a specified wrapper script or executable program that typically performs environment setup for the job, then calls the job itself, which inherits the execution environment created by the job starter. Lava controls the job starter process, rather than the job. One typical use of a job starter is to customize Lava for use with specific application environments.

Two ways to run job starters

You run job starters two ways in Lava. You can accomplish similar things with either job starter, but their functional details are slightly different.

As a command Are user-defined. They run interactive jobs submitted using `lsrun` or `lsgrun`. Command-level job starters have no effect on batch jobs, including interactive batch jobs run with `bsub -I`.

To a queue Defined by the Lava administrator, and run batch jobs submitted to a queue defined with the JOB_STARTER parameter set. Use `bsub` to submit jobs to queues with job-level job starters.

A queue-level job starter is configured in the queue definition in `lsb.queues`. See *Inside Platform Lava* for detailed information.

Pre-execution commands are not job starters

A job starter differs from a pre-execution command. A pre-execution command must run successfully and exit before the Lava job starts. It can signal Lava to dispatch the job, but because the pre-execution command is an unrelated process, it does not control the job or affect the execution environment of the job. A job starter, however, is the process that Lava controls. It is responsible for invoking Lava, and it controls the execution environment of the job. (For information on pre-execution commands, see “Pre-Execution and Post-Execution Commands” on page 14.)

Examples

The following are some examples of job starters:

- A job starter defined as `/bin/ksh -c` causes jobs to be run under a Korn shell environment
- Setting the JOB_STARTER parameter in `lsb.queues` to `$USER_STARTER` enables users to define their own job starters by defining the environment variable USER_STARTER
- Setting a job starter to `make clean` causes the command `make clean` to be run before the user job

CHAPTER 5

Controlling Jobs

Contents

Removing, Suspending, and Resuming Jobs

Lava controls jobs dispatched to a host to enforce scheduling policies or in response to user requests. The Lava system performs the following actions on a job:

- Suspend
- Resume
- Terminate

Killing a job

The `bkill` command cancels pending batch jobs and sends signals to running jobs. By default, `bkill` sends the `SIGKILL` signal to running jobs.

Before `SIGKILL` is sent, `SIGINT` and `SIGTERM` are sent to give the job a chance to catch the signals and clean up. The signals are forwarded from `mbatchd` to `sbatchd`, which waits for the job to exit before reporting the status. Because of these delays, for a short period of time after entering the `bkill` command, `bjobs` may still report that the job is running. (For descriptions of `mbatchd` and `sbatchd`, see *Inside Platform Lava*.)

Example

To kill job 3421:

```
$ bkill 3421
Job <3421> is being terminated
```

Forcing removal of a job

If a job cannot be killed in the operating system, use `bkill -r` to force the removal of the job from Lava.

The `bkill -r` command removes a job from the system without waiting for the job to terminate in the operating system. This sends the same series of signals as `bkill` without `-r`, except for the following differences:

- The job is removed from the system immediately
- The job is marked as EXIT
- Job resources that Lava monitors are released as soon as Lava receives the first signal

Suspending a job

Run `bstop` *`job_ID`*. Your job goes into `USUSP` state if the job is already started, or into `PSUSP` state if it is pending. For example:

```
$ bstop 3421
Job <3421> is being stopped
```

suspends job 3421.

`bstop` sends the `SIGSTOP` signal for sequential jobs to the job:

`SIGSTOP` cannot be caught by user programs. The `SIGSTOP` signal can be configured with the LSB_SIGSTOP parameter in `lsf.conf`.

Resuming a job

Run `bresume` *`job_ID`*. For example:

```
$ bresume 3421
Job <3421> is being resumed
```

resumes job 3421.

Resuming a user-suspended job does not put your job into `RUN` state immediately.

- If you job was pending before the suspension, `bresume` first puts your job into `PEND` state. The job then waits to be scheduled and dispatched.
- If your job was running before the suspension, `bresume` first puts your job into `SSUSP` state. The job then waits to be scheduled and dispatched.

Requeuing and Rerunning Jobs

You can kill and requeue a job while it is running or when it is suspended. Use the brequeue command to requeue the job.

You can requeue and rerun a job if the execution host or the Lava system fails while the job is running. Submit the job with the re-runnable option to enable automatic job rerun.

Requeuing a job

You can use `brequeue` to kill a job and requeue it. When the job is requeued, it is assigned the PEND status and the job's new position in the queue is after other jobs of the same priority.

- You can only use `brequeue` on running (RUN), user-suspended (USUSP), or system-suspended (SSUSP) jobs.
- Users can only requeue their own jobs. Only root and Lava administrator can requeue jobs submitted by other users.
- You cannot use `brequeue` on interactive batch jobs

Examples

```
$ brequeue 109
```

Lava kills the job with job ID 109, and requeues it in the PEND state. If job 109 has a priority of 4, it is placed after all the other jobs with the same priority.

```
$ brequeue -u user5 45 67 90
```

Lava kills and requeues three jobs belonging to `User5`. The jobs have the job IDs 45, 67, and 90.

Submitting a rerunnable job

To enable automatic job rerun at the job level, use `bsub -r`.

If the execution host fails, Lava dispatches the job to another host. You receive a mail message informing you of the host failure and the requeuing of the job.

If the Lava system fails, Lava requeues the job when the system restarts.

Forcing Jobs to Run

A pending job can be forced to run with the `brun` command. This operation can only be performed by a Lava administrator.

You can force a job to run on a particular host, to run until completion, and other restrictions. For more information, see the `brun` command in the Platform Lava man pages.

When a job is forced to run, any other constraints associated with the job such as resource requirements or dependency conditions are ignored.

Force a job to run

Use `brun -m` *`hostname job_ID`* to force a pending job to run. You must specify the host on which the job will run. For example, the following command will force the sequential job 104 to run on `hostA`:

```
$ brun -m hostA 104
```

Moving Jobs

Moving a job to the bottom of a queue

Use `bbot` to move jobs relative to your last job in the queue.

If invoked by a regular user, `bbot` moves the selected job after the last job with the same priority submitted by the user to the queue.

If invoked by the Lava administrator, `bbot` moves the selected job after the last job with the same priority submitted to the queue.

Moving a job to the top of a queue

Use `btop` to move jobs relative to your first job in the queue.

If invoked by a regular user, `btop` moves the selected job before the first job with the same priority submitted by the user to the queue.

If invoked by the Lava administrator, `btop` moves the selected job before the first job with the same priority submitted to the queue.

Switching jobs from one queue to another

You can use the command `bswitch` to change jobs from one queue to another. This is useful if you submit a job to the wrong queue, or if the job is suspended because of queue thresholds and you would like to resume the job.

Switch a single job

Use `bswitch` to move pending and running jobs from queue to queue.

In the following example, job 5309 is switched to the `priority` queue:

```
$ bswitch priority 5309
Job <5309> is switched to queue <priority>
```

```
$ bjobs -u all
JOBID    USER   STAT   QUEUE    FROM_HOST  EXEC_HOST   JOB_NAME   SUBMIT_TIME
5308     user2   RUN   normal   hostA      hostD       /job500    Oct 23 10:16
5309     user2   RUN   priority hostA      hostB       /job200    Oct 23 11:04
5311     user2   PEND  night    hostA                  /job700    Oct 23 18:17
5310     user1   PEND  night    hostB                  /myjob     Oct 23 13:45
```

Switch all jobs

Use `bswitch -q` *`from_queue to_queue`* `0` to switch all the jobs in a queue to another queue. The example below selects jobs from the `night` queue and switches them to the `idle` queue.

The `-q` option is used to operate on all jobs in a queue. The job ID number 0 specifies that all jobs from the night queue should be switched to the idle queue:

```
$ bswitch -q night idle 0
Job <5308> is switched to queue <idle>
Job <5310> is switched to queue <idle>
```

CHAPTER 6

Monitoring Jobs

Contents

Viewing Job Information

The `bjobs` command displays the status of jobs in the Lava system. For more details on these or other `bjobs` options, see the `bjobs` command in the Platform Lava man pages.

The `bjobs` command reports the status of Lava jobs.

When no options are specified, `bjobs` displays information about jobs in the PEND, RUN, USUSP, PSUSP, and SSUSP states for the current user.

For example:

```
$ bjobs
JOBID USER   STAT   QUEUE     FROM_HOST EXEC_HOST JOB_NAME    SUBMIT_TIME
3926  user1  RUN    priority  hostf     hostc     verilog     Oct 22 13:51
605   user1  SSUSP  idle      hostq     hostc     Test4       Oct 17 18:07
1480  user1  PEND   priority  hostd               generator   Oct 19 18:13
7678  user1  PEND   priority  hostd               verilog     Oct 28 13:08
7679  user1  PEND   priority  hosta               coreHunter  Oct 28 13:12
7680  user1  PEND   priority  hostb               myjob       Oct 28 13:17
```

All jobs

`bjobs -a` displays the same information as `bjobs` and in addition displays information about recently finished jobs (PEND, RUN, USUSP, PSUSP, SSUSP, DONE and EXIT statuses).

All your jobs that are still in the system and jobs that have recently finished are displayed.

Running jobs

`bjobs -r` displays information only for running jobs (RUN state).

All jobs for all users

Run `bjobs -u all` to display all jobs for all users. Job information is displayed in the following order:

1. Running jobs
2. Pending jobs in the order in which they will be scheduled
3. Jobs in high priority queues are listed before those in lower priority queues

For example:

```
$ bjobs -u all
JOBID   USER    STAT    QUEUE     FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
1004    user1   RUN     short     hostA       hostA       job0       Dec 16 09:23
1235    user3   PEND    priority  hostM                   job1       Dec 11 13:55
1234    user2   SSUSP   normal    hostD       hostM       job3       Dec 11 10:09
1250    user1   PEND    short     hostA                   job4       Dec 11 13:59
```

Jobs for specific users

Run `bjobs -u user_name` to display jobs for a specific user. For example:

```
$ bjobs -u user1
JOBID   USER    STAT    QUEUE     FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
2225    user1   USUSP   normal    hostA                   job1       Nov 16 11:55
2226    user1   PSUSP   normal    hostA                   job2       Nov 16 12:30
Detailed job information
```

`bjobs -l` with a job ID displays all the information about a job, including:

- Submission parameters
- Execution environment
- Resource usage

For example:

```
$ bjobs -l 7678
Job Id <7678>, User <user1>, Status <PEND>, Queue <priority>, Command <verilog>
Mon Oct 28 13:08:11: Submitted from host <hostD>,CWD <$HOME>,
Requested Resources <type==any && swp>35>;
PENDING REASONS:
Queue's resource requirements not satisfied:3 hosts;
Unable to reach slave lsbatch server: 1 host;
Not enough job slots: 1 host;

SCHEDULING PARAMETERS:
           r15s   r1m  r15m   ut    pg    io   ls  it  tmp  swp  mem
loadSched  -      0.7  1.0    -     4.0   -    -   -   -    -    -
loadStop   -      1.5  2.5    -     8.0   -    -   -   -    -    -
```

Pending jobs and reasons

`bjobs -p` displays information for pending jobs (PEND state) and their reasons. There can be more than one reason why the job is pending.

For example:

```
$ bjobs -p
JOBID USER   STAT   QUEUE    FROM_HOST    JOB_NAME    SUBMIT_TIME
7678  user1  PEND   priority  hostD       verilog      Oct 28 13:08
Queue's resource requirements not satisfied:3 hosts;
Unable to reach slave lsbatch server: 1 host;
Not enough job slots: 1 host;
```

The pending reasons also mention the number of hosts for each condition.

You can view reasons why a job is pending or in suspension for all users by combining the `-p` and `-u all` options.

Suspended jobs and reasons

`bjobs -s` displays information for suspended jobs (PSUSP, SSUSP, and USUSP, status) and their reasons. For example:

```
$ bjobs -s
JOBID USER  STAT  QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
605   user1 SSUSP idle  hosta     hostc     Test4    Oct 17 18:07
The host load exceeded the following threshold(s):
Paging rate: pg;
Idle time: it;
```

Viewing Job History

Sometimes you want to know what has happened to your job since it was submitted. The `bhist` command displays a summary of the pending, suspended, and running time of jobs for the user who invoked the command. Use `bhist -u all` to display a summary for all users in the cluster.

For more details on `bhist` options, see the `bhist` command in the Platform Lava man pages.

bhist does not display information about the compute hosts.

Detailed job history

The `-l` option of `bhist` displays the time information and a complete history of scheduling events for each job.

```
$ bhist -l 1531
JobId <1531>, User <user1>, Project <default>, Command< example200>
Fri Dec 27 13:04:14: Submitted from host <hostA> to Queue <priority>,
CWD <$HOME>, Specified Hosts <hostD>;
Fri Dec 27 13:04:19: Starting (Pid 8920);
Fri Dec 27 13:04:20: Running with execution home </home/user1>, Execution CWD
</home/user1>, Execution Pid <8920>;
Fri Dec 27 13:05:49: Suspended by the user or administrator;
Fri Dec 27 13:05:56: Suspended: Waiting for re-scheduling after being resumed
by user;
Fri Dec 27 13:05:57: Running;
Fri Dec 27 13:07:52: Done successfully. The CPU time used is 28.3 seconds.

Summary of time in seconds spent in various states by Sat Dec 27 13:07:52 1997
PEND  PSUSP  RUN  USUSP  SSUSP  UNKWN  TOTAL
5     0      205  7      1      0      218
```

History of jobs not listed in active event log

Lava periodically backs up and trims the job history log. By default, `bhist` only displays job history from the current event log file. You can use `bhist -n num_logfiles` to display the history for jobs that completed some time ago and are no longer listed in the active event log.

bhist -n *num_logfiles*

The `-n num_logfiles` option tells the `bhist` command to search through the specified number of log files instead of only searching the current log file.

Log files are searched in reverse time order. For example, the command `bhist -n 3` searches the current event log file and then the two most recent backup files.

Examples

`bhist -n 1`	searches the current event log file `lsb.events`
`bhist -n 2`	searches `lsb.events` and `lsb.events.1`
`bhist -n 3`	searches `lsb.events`, `lsb.events.1`, `lsb.events.2`
`bhist -n 0`	searches all event log files in LSB_SHAREDIR (For a description of LSB_SHAREDIR, see *Inside Platform Lava*.)

Viewing Job Output

The output from a job is normally not available until the job is finished. However, Lava provides the `bpeek` command for you to look at the output the job has produced so far.

By default, `bpeek` shows the output from the most recently submitted job. You can also select the job by queue or execution host, or specify the job ID or job name on the command line.

For more details on `bpeek` options, see the `bpeek` command in the Platform Lava man pages.

Output of a running job

Only the job owner can use `bpeek` to see job output. The `bpeek` command will not work on a job running under a different user account.

To save time, you can use this command to check if your job is behaving as you expected and kill the job if it is running away or producing unusable results.

For example:

```
$ bpeek 1234
<< output from stdout >>
Starting phase 1
Phase 1 done
Calculating new parameters
...
```

Index