ScriptEngine HPC Task Package

Uwe Fladrich

Apr 28, 2023

CONTENTS:

CHAPTER

ONE

INTRODUCTION

The ScriptEngine HPC Task package (SE HPC Tasks).

CHAPTER

TWO

SLURM TASKS

Support for the SLURM workload manager.

2.1 The `hpc.slurm.sbatch` task

This task allows to send ScriptEngine jobs to SLURM queues by providing the functionallity of the SLURM `sbatch` command to ScriptEngine scripts. The usage pattern is:

```
- hpc.slurm.sbatch:
    scripts: <SE_SCRIPT | LIST_OF_SE_SCRIPTS>  # optional
    hetjob_spec: <LIST_OF_SBATCH_OPTIONS>  # optional
    submit_from_sbatch: <true | false>  # optional, default False
    stop_after_submit:  <true | false>  # optional, default True
    set_jobid: <CONTEXT_NAME>  # optional

    <SBATCH_OPTIONS>  # optional
```

The main usage principle of `hpc.slurm.sbatch` is that a new batch job is created and sent into a SLURM queue. Once SLURM executes the job, one or more ScriptEngine scripts are run.

There are two ways to specify which scripts are run in the batch job. By default (no `script` argument is given), the batch job runs the script(s) given on the `se` command line. For example, if the following script (assumed name `sbatch.yml`):

```
- hpc.slurm.sbatch:
    account: <MY_SLURM_ACCOUNT>
    time: !noparse 0:10:00

- base.echo:
    msg: Hello world, from batch job!
```

is run with `se sbatch.yml`, a batch job will be queued, which eventually writes "Hello world, from batch job!" to the default job logfile. Using this default will be the desired behavior in most use cases. However, it is possible to have the batch job run a different script (or scripts) and not the initiall one, by specifying one or more other ScriptEngine scripts with the `scripts` arguments. More than one scripts have to specified as a list.

Most of the `hpc.slurm.sbatch` arguments will be passed right through to the `sbatch` command. Thus, in the above example, the command executed under the hood is:

```
sbatch --account MY_SLURM_ACCOUNT --time 0:10:00 se sbatch.yml
```

Only few arguments are processed by the `hpc.slurm.sbatch` task itself, see the usage pattern above. Thus, it is possible to use any `sbatch` argument, as long as they are valid long arguments (i.e. with the double dash syntax). Note that *no checking is done* for validity of the `sbatch` arguments and options!

An important principle of `hpc.slurm.sbatch` is that on the initial execution, it will stop the processing of the current script once the batch job is queued. Hence, when the above example script is run, a job is put in the batch queue (first task), but the `base.echo` task is not executed. When the script is run (again) from within the batch job, the `hpc.slurm.sbatch` task detects that it is in a batch job and does nothing. Therefore, the following echo task is run as part of the job.

Again, this behavior will be appropriate in most use cases. The script is run until the sbatch task, a job is queued and processing stops. Once the job is running, `hpc.slurm.sbatch` does nothing and all other tasks are run.

Sometimes, though, it makes sense to submit a batch job even if the current script already runs in a batch job itself. For example, one may want to queue a follow-on job at the end of the script. In order to do this, one needs to set:

```
- hpc.slurm.sbatch:
    [...]
    submit_from_sbatch: true
    [...]
```

If `submit_from_sbatch` is set to `true` a new job is queued, even if the current script is itself running in a batch job on its own.

A related switch is `stop_after_submit`, which defaults to `True`. If it is set to `False` the script will continue after a new SLURM job was submitted. If `stop_after_submit` is not explicitly set (or set to `True`) the script execution will be stopped, as described above.

2.2 Saving the SLURM JOBID

When the job submission via SLURM sbatch succeeds, it is possible to save the JOBID of the new job in the ScriptEngine context. For this, the `set_jobid` task argument can be set to a key for the context dictionary. If `set_jobid` is not given (or set to `False`), the JOBID is not stored in the context. Note that only simple context keys, no dot-separated values, are supported.

Example:

```
- hpc.slurm.sbatch:
    [...]
    set_jobid: jobid
    [...]
- base.echo:
    msg: "Submitted job with ID {{jobid}}."
```

2.3 SLURM Heterogeneous Job Support

The `hpc.slurm.sbatch` task support submitting heterogeneous SLURM jobs by providing the `hetjob_spec` option:

```
- hpc.slurm.sbatch:
    - time: 10
    - hetjob_spec:
        - nodes: 1
```

(continues on next page)

(continued from previous page)

```
        - nodes: 2

- base.command:
    name: srun
    args: [
        -l,
        --ntasks, 1, /usr/bin/hostname, ':',
        --ntasks, 10, --ntasks-per-node, 5, /usr/bin/hostname
    ]
```

In this example, a heterogeneous job with two components is submitted to SLURM, the first requesting one node and the second two nodes. The `srun` command in the second task of the script starts executables on this allocated nodes while specifying further job characteristics (such as the number of tasks and tasks per node).

The `hetjob_spec` argument takes a list of dictionaries and passes the keys of each dictionary on to `sbatch` as specification for each respective component of the heterogeneous job. Note that in the example above, each dictionary contains only one key-value pair, the number of requested nodes.

CHAPTER

THREE

ENVIRONMENT MODULE TASKS

The ScriptEngine HPC Task package allows interaction with environment modules, often used on HPC systems to configure the user's environment for installed software packages. This task package supports the two most common module implementations: *Lmod* (https://lmod.readthedocs.io) and *Environment Modules* (https://modules.readthedocs.io).

The ScriptEngine tasks in this package allow modules to be loaded or unloaded in SE scripts and can thus modify the environment and available software during the execution of scripts.

3.1 Prerequisites

A fairly recent version of either *Lmod* (source code at https://github.com/TACC/Lmod) or *Environment modules* (http://modules.sourceforge.net) is needed. In particular, the module version should provide Python3 initialisation scripts.

If, however, the module version installed on an HPC system does not provide Python3 init scripts, it is possible to initialise the tasks from a user-provided initialisation script. This allows to use the ScriptEngine module tasks even on systems with an outdated module system. See *The init argument* below.

3.2 The `hpc.module` task

Runs any module command.

Usage:

```
- hpc.module:
    cmd: <COMMAND_NAME>
    args: <LIST_OF_ARGS>  # optional
```

Note that *no checking is done* as to wether the command and arguments are valid! In particular, there is no guarantee that the command will run given the particular module implementation (Environment modules or Lmod) and the version installed on the HPC system. The command name and arguments are passed to the underlying module system and runtime errors reported via ScriptEngine.

For example:

```
- hpc.module:
    cmd: list
```

will run `module list` and write the result to standard output.

If the module command requires arguments, they are given via the task argument `args`. The arguments have to be specified as a list (even if there is only one):

```
- hpc.module:
    cmd: show
    args: [ gcc/10.2 ]
```

3.3 The `hpc.module.load` task

This task is provided for convenience, as it allows for a shorter syntax to load modules (compared to the `hpc.module` task using `cmd: load`)

Usage:

```
- hpc.module.load:
    names: <MODULE_NAME | LIST_OF_MODULE_NAMES>
```

Examples:

```
- hpc.module.load:
    names:
    - gcc/10.2
    - netcdf/4.3.0
```

If there is only a single module to be loaded, the name can be given without using a list:

```
- hpc.module.load:
    names: git/2.19.3
```

3.4 The `init` argument

As mentioned under *Prerequisites*, the `hpc.module` tasks need Python3 initialisation scripts, usually provided by recent versions of the module systems. Sometimes, however, older module versions are installed on some HPC systems and Python3 support is missing. What the `hpc.module` scripts really need is an initialisation script, the rest of the implementation usually works fine even with older module versions. Hence, it is possible to manually provide the initialisation scripts.

In order to provide a user defined initialisation script, the `init` argument can be added to any of the module tasks (`hpc.module` or `hpc.module.load`). Since the initialisation is only done once, the `init` argument is only needed at the *first task executed*. If the `init` argument is present at any subsequent task, it is ignored. If the `init` argument is missing at the first executed task (and default initialisation does not work) initialisation will fail.

The `init` argument must specify the path at which the initialisation script can be found, for example:

```
- hpc.module.load:
    init: /home/user/lmod/init/env_modules_python.py
    names: gcc/10.2

- hpc.module:
    cmd: list
```

In order to follow which task is initialising the module system and from what location, run ScriptEngine with `se --loglevel debug [...]`.

CHAPTER

FOUR

INDICES AND TABLES

- genindex