Getting Started with Parallel Computing using MATLAB on the Magnitude HPC Cluster – Remote Submission

This document provides the steps to configure MATLAB to submit jobs to a cluster, retrieve results, and debug errors.

INSTALLATION and CONFIGURATION

The Magnitude MATLAB support package can be found as follows

Windows: https://www.uni-due.de/~hk0798/matlab/ude.nonshared.R2020b.zip
Linux/macOS: https://www.uni-due.de/~hk0798/matlab/ude.nonshared.R2020b.tar.gz

Download the appropriate archive file and start MATLAB. The archive file should be untarred/unzipped in the location returned by calling

```
>> userpath
```

Configure MATLAB to run parallel jobs on your cluster by calling `configCluster`. `configCluster` only needs to be called <u>once</u> per version of MATLAB.

```
>> configCluster
```

Submission to the remote cluster requires SSH credentials. You will be prompted for your ssh username and password or identity file (private key). The username and location of the private key will be stored in MATLAB for future sessions.

Jobs will now default to the cluster rather than submit to the local machine.

NOTE: If you would like to run the pool of workers on the same machine running MATLAB, simply run the following command and skip the remaining sections of this document.

```
>> % Get a handle to the local resources
>> c = parcluster('local');
```

CONFIGURING JOBS

Prior to submitting the job, we can specify various parameters to pass to our jobs, such as queue, e-mail, walltime, etc. *None of these are required*.

```
>> % Get a handle to the cluster
>> c = parcluster;

>> % Specify an account to use for MATLAB jobs
>> c.AdditionalProperties.AccountName = 'account-name';

>> % Request a FAT node
>> c.AdditionalProperties.Constraint = 'FAT';
```

```
>> % Specify e-mail address to receive notifications about your job
>> c.AdditionalProperties.EmailAddress = 'user-id@uni-due.de';

>> % Specify memory to use for MATLAB jobs, per core (MB)
>> c.AdditionalProperties.MemUsage = '4000';

>> % Specify a queue to use for MATLAB jobs
>> c.AdditionalProperties.QueueName = 'queue-name';

>> % Request reservation
>> c.AdditionalProperties.Reservation = 'name-of-reservation';

>> % Wait up to 30 minutes for a single switch; otherwise, proceed
>> c.AdditionalProperties.Switches = '1@00:30:00';

>> % Specify the walltime (e.g. 5 hours)
>> c.AdditionalProperties.WallTime = '05:00:00';
```

Save changes after modifying AdditionalProperties for the above changes to persist between MATLAB sessions.

```
>> c.saveProfile
```

To see the values of the current configuration options, display AdditionalProperties.

```
>> % To view current properties
>> c.AdditionalProperties
```

Unset a value when no longer needed.

```
>> % Turn off email notifications
>> c.AdditionalProperties.EmailAddress = '';

>> c.saveProfile
```

INDEPENDENT BATCH JOB

Use the `batch` command to submit asynchronous jobs to the cluster. The `batch` command will return a job object which is used to access the output of the submitted job. See the MATLAB documentation for more help on `batch`.

```
>> % Get a handle to the cluster
>> c = parcluster;

>> % Submit job to query where MATLAB is running on the cluster
>> j = c.batch(@pwd, 1, {}, …
        'CurrentFolder','.', 'AutoAddClientPath',false);

>> % Query job for state
```

```
>> j.State

>> % If state is finished, fetch the results
>> j.fetchOutputs{:}

>> % Delete the job after results are no longer needed
>> j.delete
```

To retrieve a list of currently running or completed jobs, call `parcluster` to retrieve the cluster object. The cluster object stores an array of jobs that were run, are running, or are queued to run. This allows us to fetch the results of completed jobs. Retrieve and view the list of jobs as shown below.

```
>> c = parcluster;
>> jobs = c.Jobs;
```

Once we've identified the job we want, we can retrieve the results as we've done previously.

`fetchOutputs` is used to retrieve function output arguments; if calling `batch` with a script, use `load` instead.   Data that has been written to files on the cluster needs be retrieved directly from the file system (e.g. via ftp).

To view results of a previously completed job:

```
>> % Get a handle to the job with ID 2
>> j2 = c.Jobs(2);
```

NOTE: You can view a list of your jobs, as well as their IDs, using the above `c.Jobs` command.

```
>> % Fetch results for job with ID 2
>> j2.fetchOutputs{:}
```

PARALLEL BATCH JOB

Users can also submit parallel workflows with the `batch` command. Let's use the following example for a parallel job, which is saved as `parallel_example.m`.

```matlab
function t = parallel_example(iter)

if nargin==0, iter = 8; end

disp('Start sim')

t0 = tic;
parfor idx = 1:iter
    A(idx) = idx;
    pause(2)
end
t = toc(t0);

disp('Sim Completed')
```

This time when we use the `batch` command, to run a parallel job, we'll also specify a MATLAB Pool.

```
>> % Get a handle to the cluster
>> c = parcluster;

>> % Submit a batch pool job using 4 workers for 16 simulations
>> j = c.batch(@parallel_example, 1, {16}, 'Pool',4, …
        'CurrentFolder','.', 'AutoAddClientPath',false);

>> % View current job status
>> j.State

>> % Fetch the results after a finished state is retrieved
>> j.fetchOutputs{:}

ans =

8.8872
```

The job ran in 8.89 seconds using four workers. Note that these jobs will always request N+1 CPU cores, since one worker is required to manage the batch job and pool of workers.   For example, a job that needs eight workers will consume nine CPU cores.

We'll run the same simulation but increase the Pool size. This time, to retrieve the results later, we'll keep track of the job ID.

NOTE: For some applications, there will be a diminishing return when allocating too many workers, as the overhead may exceed computation time.

```
>> % Get a handle to the cluster
>> c = parcluster;

>> % Submit a batch pool job using 8 workers for 16 simulations
>> j = c.batch(@parallel_example, 1, {16}, 'Pool', 8, …
        'CurrentFolder','.', 'AutoAddClientPath',false);

>> % Get the job ID
>> id = j.ID

id =

4

>> % Clear j from workspace (as though we quit MATLAB)
>> clear j
```

Once we have a handle to the cluster, we'll call the `findJob` method to search for the job with the specified job ID.

```
>> % Get a handle to the cluster
>> c = parcluster;

>> % Find the old job
>> j = c.findJob('ID', 4);

>> % Retrieve the state of the job
>> j.State

ans =

    finished

>> % Fetch the results
>> j.fetchOutputs{:};

ans =

    4.7270
```
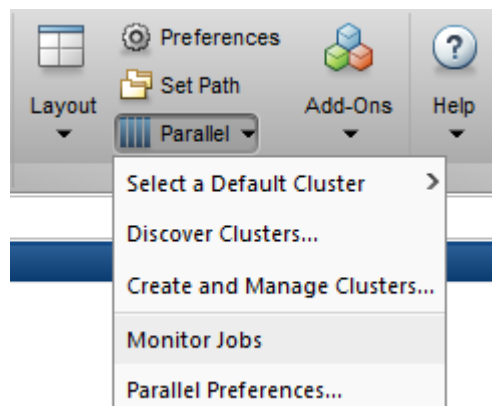
The job now runs in 4.73 seconds using eight workers. Run code with different number of workers to determine the ideal number to use.

Alternatively, to retrieve job results via a graphical user interface, use the Job Monitor (Parallel > Monitor Jobs).

DEBUGGING

If a serial job produces an error, call the `getDebugLog` method to view the error log file. When submitting independent jobs, with multiple tasks, specify the task number.

```
>> c.getDebugLog(j.Tasks(3))
```

For Pool jobs, only specify the job object.

```
>> c.getDebugLog(j)
```

When troubleshooting a job, the cluster admin may request the scheduler ID of the job. This can be derived by calling `schedID`

```
>> schedID(j)

ans =

     25539
```

TO LEARN MORE

To learn more about the MATLAB Parallel Computing Toolbox, check out these resources:

- [Parallel Computing Coding Examples](#)
- [Parallel Computing Documentation](#)
- [Parallel Computing Overview](#)
- [Parallel Computing Tutorials](#)
- [Parallel Computing Videos](#)
- [Parallel Computing Webinars](#)