Getting Started with Parallel Computing using MATLAB (R2022a) on the PARAM Ganga HPC Cluster

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

INSTALLATION and CONFIGURATION

The PARAM Ganga MATLAB support package can be found as follows

Windows: (Click Here do Download)
Linux/macOS: (Click Here do Download)

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 once 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 submit to the local machine then run the following command:

```
>> % 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. The following is a partial list of parameters. See `AdditionalProperties` for the complete list. *None of these are required. Refer to the User Manual for more information.*

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

>> % Specify the account to use
>> c.AdditionalProperties.AccountName = 'account-name';

>> % Request a constraint
>> c.AdditionalProperties.Constraint = 'constraint-name';
```

```
>> % Request email notification of job status
>> c.AdditionalProperties.EmailAddress = 'user-id@iitr.ac.in';

>> % Specify number of GPUs to use
>> c.AdditionalProperties.GpusPerNode = 1;
>> c.AdditionalProperties.GpuCard = 'gpu-card';

>> % Specify memory to use, per core
>> c.AdditionalProperties.MemUsage = '6gb';

>> % Specify the QOS
>> c.AdditionalProperties.QoS = 'qos-name';

>> % Specify the queue to use (default: highmemory)
>> % Note: Each queue has a different minimum number of required cores
>> c.AdditionalProperties.QueueName = 'queue-name';

>> % Request to run on exclusive node(s) (default: false)
>> c.AdditionalProperties.RequireExclusiveNode = true;

>> % Specify reservation to use
>> c.AdditionalProperties.Reservation = 'reservation-name';

>> % Specify the wall time (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
```

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`.

```
function [t, A] = parallel_example(iter)

if nargin==0
    iter = 240;
end

disp('Start sim')

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

disp('Sim completed')

save RESULTS A

end
```

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 24 workers for 480 simulations
>> job = c.batch(@parallel_example, 1, {480}, 'Pool',24, ...
       'CurrentFolder','.');

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

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

ans =

     41.5629
```

The job ran in 41.56 seconds using 24 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 24 workers will consume 25 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.

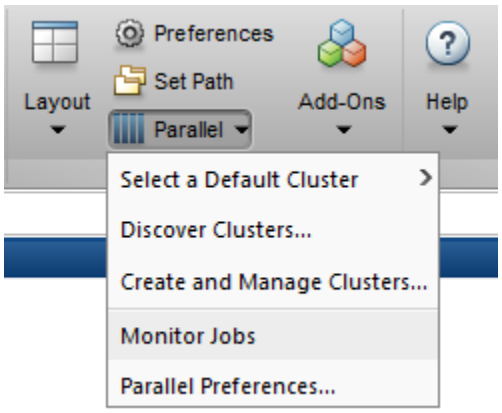
```
>> % Get a handle to the cluster
>> c = parcluster;

>> % Submit a batch pool job using 8 workers for 16 simulations
>> job = c.batch(@parallel_example, 1, {480}, 'Pool', 48, ...
       'CurrentFolder','.');

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

id =

     4

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

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
>> job = c.findJob('ID', 4);

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

ans =

     finished

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

ans =

     20.6194
```

The job now runs in 20.6 seconds using 48 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 job produces an error, call the `getDebugLog` method to view the error log file.

```
>> c.getDebugLog(job)
```

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

```
>> job.getTaskSchedulerIDs

ans =

      55497
```

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