Getting Started with Parallel Computing using MATLAB on the Hopper HPC Cluster

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

**CONFIGURATION – MATLAB client on the cluster**

After logging into the cluster, start MATLAB. On the Home tab, click Parallel > Discover Clusters… to discover the profile.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

ALTERNATE:

After logging into the cluster, start MATLAB. Create a cluster profile to run parallel jobs on the cluster by calling configCluster. configCluster should only be called once per cluster.

>> configCluster

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

**NOTE**: To submit to the local machine instead of the cluster, run the following:

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

**INSTALLATION and CONFIGURATION – MATLAB client on the desktop**

The Hopper MATLAB support package can be found at TBD.

Download the ZIP file and start MATLAB. The ZIP file should be unzipped in the location returned by calling

>> userpath

Configure MATLAB to run parallel jobs on the cluster by calling configCluster (configCluster must be in the *userpath* folder). configCluster only needs to be called once per version of MATLAB.

>> configCluster

Submission to the cluster requires SSH credentials. You will be prompted for 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**: To submit to the local machine instead of the cluster, run the following:

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

**CONFIGURING JOBS**

Prior to submitting the job, various parameters can be assigned, such as queue, e-mail, walltime, etc. The following is a partial list of parameters. See AdditionalProperties for the complete list.

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

[**REQUIRED**]

>> % Specify memory to use, per core (default: 4gb)

>> c.AdditionalProperties.MemPerCPU = '6gb';

[**OPTIONAL**]

>> % Specify the account

>> c.AdditionalProperties.AccountName = 'account-name';

>> % Specify a constraint

>> c.AdditionalProperties.Constraint = 'feature-name';

>> % Request email notification of job status

>> c.AdditionalProperties.EmailAddress = 'user-id@vassar.edu';

>> % Specify number of GPUs (default: 0)

>> c.AdditionalProperties.GPUsPerNode = 1;

>> c.AdditionalProperties.GPUCard = 'gpu-card';

>> % Specify the partition

>> c.AdditionalProperties.Partition = 'partition-name';

>> % Specify cores per node (default: 0)  
>> c.AdditionalProperties.ProcsPerNode = 4;

>> % Set node exclusivity (default: false)  
>> c.AdditionalProperties.RequireExclusiveNode = true;

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

>> % Specify the wall time (e.g., 1 day, 5 hours, 30 minutes)

>> c.AdditionalProperties.WallTime = '1-05:30';

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

**INTERACTIVE JOBS - MATLAB client on the cluster**

To run an interactive pool job on the cluster, continue to use parpool as before.

>> % Get a handle to the cluster

>> c = parcluster;

>> % Open a pool of 64 workers on the cluster

>> pool = c.parpool(64);

Rather than running local on the local machine, the pool can now run across multiple nodes on the cluster.

>> % Run a parfor over 1000 iterations

>> parfor idx = 1:1000

a(idx) = rand;

end

Delete the pool when it’s no longer needed.

>> % Delete the pool

>> pool.delete

**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

>> job = c.batch(@pwd, 1, {}, 'CurrentFolder','.');

>> % Query job for state

>> job.State

>> % If state is finished, fetch the results

>> job.fetchOutputs{:}

>> % Delete the job after results are no longer needed

>> job.delete

To retrieve a list of running or completed jobs, call parcluster to return the cluster object. The cluster object stores an array of jobs that are queued to run, are running, have run, or have failed. Retrieve and view the list of jobs as shown below.

>> c = parcluster;

>> jobs = c.Jobs

>>

>> % Get a handle to the second job in the list

>> job2 = c.Jobs(2);

Once the job has been selected, fetch the results as previously done.

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 sftp).

>> % Fetch all results from the second job in the list

>> job2.fetchOutputs{:}

**PARALLEL BATCH JOB**

batch can also submit parallel workflows. Let’s use the following example for a parallel job, which is saved as parallel\_example.m.

|  |
| --- |
| **function [sim\_t, A] = parallel\_example(iter)**    **if nargin==0**  **iter = 8;**  **end**    **disp('Start sim')**    **t0 = tic;**  **parfor idx = 1:iter**  **A(idx) = idx;**  **pause(2)**  **idx**  **end**  **sim\_t = toc(t0);**    **disp('Sim completed')**    **save RESULTS A**    **end** |

This time when using the batch command, also specify a MATLAB Pool argument.

>> % Get a handle to the cluster

>> c = parcluster;

>> % Submit a batch pool job using 4 workers for 16 simulations

>> job = c.batch(@parallel\_example, 1, {16}, 'Pool',4, ...

'CurrentFolder','.');

>> % View current job status

>> job.State

>> % Fetch the results after a finished state is retrieved

>> job.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 request nine CPU cores.

Run the same simulation but increase the Pool size. This time, to retrieve the results later, 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, {16}, 'Pool',8, ...

'CurrentFolder','.');

>> % Get the job ID

>> id = job.ID

id =

4

>> % Clear job from workspace (as though MATLAB exited)

>> clear job

With a handle to the cluster, the findJob method searches 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 =

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).

A screenshot of a computer

Description automatically generated

**HELPER FUNCTIONS**

|  |  |  |
| --- | --- | --- |
| Function | Description | Applies Only to Desktop |
| clusterFeatures | List of cluster features/constraints |  |
| clusterGpuCards | List of cluster GPU cards |  |
| clusterPartitionNames | List of cluster partition/queue names |  |
| disableArchiving | Modify file archiving to resolve file mirroring issue | true |
| fixConnection | Reestablish cluster connection (e.g., after reconnection of VPN) | true |
| willRun | Explain why job is queued |  |

**DEBUGGING**

If a serial job produces an error, call the getDebugLog method to view the error log file. When submitting an independent job, specify the task.

>> c.getDebugLog(job.Tasks)

For Pool jobs, only specify the job object.

>> 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 (call schedID(job) before R2019b).

>> job.getTaskSchedulerIDs()

ans =

25539

**TO LEARN MORE**

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

* [Parallel Computing Overview](http://www.mathworks.com/products/parallel-computing/index.html)
* [Parallel Computing Documentation](http://www.mathworks.com/help/distcomp/index.html)
* [Parallel Computing Coding Examples](https://www.mathworks.com/help/parallel-computing/examples.html)
* [Parallel Computing Tutorials](http://www.mathworks.com/products/parallel-computing/tutorials.html)
* [Parallel Computing Videos](http://www.mathworks.com/products/parallel-computing/videos.html)
* [Parallel Computing Webinars](http://www.mathworks.com/products/parallel-computing/webinars.html)