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

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

**CONFIGURATION**

After logging into the cluster, start MATLAB. Configure MATLAB to run parallel jobs on your cluster by calling generateNetbatchProfile.

>> rehash toolboxcache

>> % Initial Netbatch profile with Remote Queue and Remote Qslot

>> generateNetbatchProfile('pdx\_abc','/ptm/def');

**CONFIGURING JOBS**

Prior to submitting the job, we can specify various parameters to pass to our jobs, such as memory per CPU, etc. Only RemoteQslot and RemoteQueue are required (which are already done when calling generateNetbatchProfile).

>> % Get a handle to the cluster  
>> c = parcluster('netbatch');

[**REQUIRED**]

>> % Specify the Remote Qslot

>> c.AdditionalProperties.RemoteQslot = 'remote-qslot-name';

>> % Specify the Remote Queue

>> c.AdditionalProperties.RemoteQueue = 'remote-queue-name';

[**OPTIONAL**]

>> % Request a specific machine class

>> c.AdditionalProperties.MachineClass = 'machine-class-name';

>> % Specify memory to use (in GB) per core

>> % (default: 0, i.e., all memory)

>> c.AdditionalProperties.MemPerCpu = 6;

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

**INTERACTIVE JOBS**

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

>> % Get a handle to the cluster

>> c = parcluster('netbatch');

>> % 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) = …

end

Once we’re done with the pool, delete it.

>> % 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('netbatch');

>> % Submit job to query where MATLAB is running on the cluster

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

>> % 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 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('netbatch');

>> 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.

To view results of a previously completed job:

>> % Get a handle to the job with ID 2

>> job2 = 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

>> job2.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.

|  |
| --- |
| **function [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**  **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('netbatch');

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

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

>> % 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 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

>> job = c.batch(@parallel\_example, 1, {16}, 'Pool', 8);

>> % 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

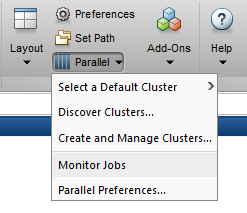
>> job.fetchOutputs{:};

ans =

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).



**LICENSING**

Ensure that you are licensed to use MATLAB Parallel Server on your Netbatch cluster.

**DEBUGGING**

If a parallel pool fails to start, call the following example.

>> pctconfig('preservejobs',true);

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

>> pool = c.parpool(64);

>> c.getDebugLog(c.Jobs(end))

>> c.Jobs(end).delete

If a serial batch job produces an error, call the getDebugLog method to view the error log file.

>> c.getDebugLog(job.Tasks)

For parallel batch 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

>> job.getTaskSchedulerIDs{:}

ans =

'70'

**TO LEARN MORE**

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

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