MPI-Cluster

Available GPUs

To find out which Machines have GPUs installed you can use:

```
condor_status -constraint 'PartitionableSlot && TotalGpus > 0' \ -af:h
Machine TotalGPUs TotalCpus CUDADeviceName TotalMemory
CUDAGlobalMemoryMb CUDACapability
```

Some infos about the available compute nodes (last updated in February 2024, we recently lost the V100-PCIE gpus):

	CUDADeviceName	CUDACapability	CUDAGlobalMemoryMb	TotalCpus	TotalMem
RTX 2080	NVIDIA GeForce RTX 2080 Ti	7.5	11 GB	28	772650
RTX 6000	Quadro RTX 6000	7.5	24 GB	36 or 128	772652
V100	Tesla V100-SXM2-32GB	7.0	32 GB	28	772657
~~V100~~	~~Tesla V100-PCIE-32GB~~	7.0	32 GB		515855
~~V100~~	~~Tesla V100-PCIE-16GB~~	7.0	16 GB		515855
A100	NVIDIA A100-SXM4-40GB	8.0	40 GB	128	1031860 (1TB)
A100	NVIDIA A100-SXM4-80GB	8.0	80 GB		2051956 (2TB)
A100	NVIDIA A100-SXM4-80GB	8.0	80 GB	96	2051955 (2TB)
H100	NVIDIA H100	9.0	95 GB		1031719 (1TB)
H100	NVIDIA H100 80GB HBM3	9.0	80 GB	192	3096017

- `CUDAGlobalMemoryMb` refers to the total amount of global memory available on a single GPU device. `CUDAGlobalMemoryMb` is specific to individual GPU devices rather than the entire node

- `TotalMemory` total physical memory (RAM) available on a machine/node in the Condor pool. It represents the combined amount of RAM across all CPU and GPU devices on that particular machine
- `TotalCpus` number of CPUs per machine

To see, which of these GPUs are currently available:

```
condor_status -constraint 'PartitionableSlot && Gpus > 0' -af:h Machine
TotalGpus GPUs CUDADeviceName Cpus Memory/1024
```

You can also check the status of a subset of GPUs:

```
condor_status -constraint 'CUDADeviceName=="Tesla V100-PCIE-16GB"'
condor_status -constraint 'CUDADeviceName=="Tesla V100-SXM2-32GB"'
condor_status -constraint 'CUDADeviceName=="NVIDIA A100-SXM4-40GB"' -
constraint 'PartitionableSlot && Gpus > 0' condor_status -constraint
'CUDADeviceName=="NVIDIA A100-SXM4-80GB"' -constraint
'PartitionableSlot && Gpus > 0' condor_status -constraint
'Machine=="g025.internal.cluster.is.localnet" ||
Machine=="g026.internal.cluster.is.localnet" ||
Machine=="g027.internal.cluster.is.localnet"'
```

Submit Interactive jobs

Get a CPU: `condor_submit_bid 25 -i -append request_cpus=2 -append request_memory=10000`

Get a GPU: `condor_submit_bid 25 -i -append request_cpus=2 -append request_gpus=1 -append request_memory=4096`

Get an A100: `condor_submit_bid 25 -i -append request_gpus=1 -append "requirements=TARGET.CUDACapability == 8.0"`

Get an H100: `condor_submit_bid 25 -i -append request_gpus=1 -append "requirements=TARGET.CUDACapability == 9.0"`

Get a GPU by name: `condor_submit_bid 25 -i -append request_gpus=1 -append "requirements=TARGET.CUDADeviceName==\"NVIDIA A100-SXM4-80GB\""`

Submit a job

Suppose that we want to a execute a python script with some args.
We'll proceed as follows:

1. create a shell script `run.sh` that activate the conda env and executes the python file
2. create a condor submission file `condor.sub`
3. submit the file to condor with `condor_submit_bid 25 condor.sub`
4. create a shell script `run.sh` that activate the conda env and executes the python file [it should have execute permisions, so do chmod +x]:

```
#!/bin/bash source ~/miniforge3/etc/profile.d/conda.sh # equivalent to conda init conda activate my_env # optionally parse args arg_1=$1 arg_2=$2 # execute python script python example.py --arg_1=$arg_1 --arg_2=$arg_2
```

2. `condor.sub` [working directory will be where this is launched from]

```
LOGS_DIR=/fast/najroldi/logs/ssm executable=/home/najroldi/cluster101/run.sh # Optional args (will be passed to the executable) arg_1 = code/config.yaml arg_2 = 1.0 arguments = $(arg_1) $(arg_2) # Logs # (remember to create the directory paths, including err/ out/ log/, before launching the job) error = $(LOGS_DIR)/err/job.$(Cluster).$(Process).err output = $(LOGS_DIR)/out/job.$(Cluster).$(Process).out log = $(LOGS_DIR)/log/job.$(Cluster).$(Process).log # Specs request_memory = 100000 request_cpus = 4 request_gpus = 1 requirements = (TARGET.CUDADeviceName == "NVIDIA A100-SXM4-80GB") n_jobs = 1 queue $(n_jobs)
```

(notice that there is no shebang at the top, also `executable` should be the *full path* to `run.sh`)

3. submit from the terminal: `condor_submit_bid 25 condor.sub`
4. check your job status from terminal: `condor_q USERNAME`

Submit a sweep

We can slightly modify the previous example to run a sweep.

The crucial point is that if we specify `queue $(n_jobs)` with `n_jobs > 1` in `condor.sub`, then condor will submit `n_jobs` copies of that job. The copies will be identical, apart from the `$(Process)` variable, that will range from `0` to `n_jobs-1`. The trick is then to use this variable to distinguish jobs. We hence pass `$(Process)` to the shell script, which takes care of distinguishing between jobs. Alternatively, we could pass `$(Process)` to the python script itself, and let it deal with assigning the right hyperparameter to each job in the sweep.

Example

Let's say that we want to run a sweep over 4 learning rates. Here we identify the learning rate associated with each process from the executable.

1. `run.sh`

```
#!/bin/bash source ~/miniforge3/etc/profile.d/conda.sh conda
activate my_env # lr sweep lrs=(0.01 0.03 0.1 0.3) # retrieve lr
from condor process process=$1 lr=${lrs[$process]} # execute python
script python example.py --lr=$lr
```

2. `condor.sub`:

```
LOGS_DIR=/fast/najroldi/logs/ssm
executable=/home/najroldi/cluster101/run.sh # Optional args (will
be passed to the executable) arguments = $(Process) # Logs #
(remember to create the directory paths, including err/ out/ log/,
before launching the job) error =
$(LOGS_DIR)/err/job.$(Cluster).$(Process).err output =
$(LOGS_DIR)/out/job.$(Cluster).$(Process).out log =
$(LOGS_DIR)/log/job.$(Cluster).$(Process).log # Specs
request_memory = 100000 request_cpus = 4 request_gpus = 1
requirements = (TARGET.CUDADeviceName == "NVIDIA A100-SXM4-80GB") #
since we have 4 learning rates in the sweep, we queue 4 jobs n_jobs
= 4 queue $(n_jobs)
```

Be careful if you submit a job, and modify the executable while the job is idle! If this happens, the latest version of the executable will be executed in the cluster! On the other hand, you can safely change the submission file after launching a job.

An alternative example

Jupyter Notebook in an interactive job

To run a Jupyter notebook in an interactive job, you need to

1. start a Jupyter Notebook server on an interactive job
2. create an SSH tunnel to connect to the Jupyter Notebook server

Here is a breakdown of how to proceed:

1. open a terminal on your local machine
    a. ssh to mpi: `ssh username@login.cluster.is.localnet`
    b. request and interactive job: `condor_submit_bid 100 -i -append "request_cpus=2" -append "request_gpus=1" -append "request_memory=10000”`
    c. let’s say you get a job allocation on `g109`
    d. start a Jupyter Notebook server on `g109` : `jupyter notebook --no-browser --port=7777 --ip=0.0.0.0`
        i. `--no-browser` : prevents it from trying to open a web browser on the HPC compute node
        ii. `--port=7777` : specifies the port (`7777`) that the Jupyter server will listen on
        iii. `--ip=0.0.0.0` : the server will accept connections from any IP address
2. open a second terminal on your local machine