Condor

Contents

1 Useful Links

- PSI Condor wiki: `https://wiki.esat.kuleuven.be/visics/condor`
- Condor Manual: `http://research.cs.wisc.edu/htcondor/manual/v8.4/index.html`
- DAGman manual: `http://research.cs.wisc.edu/htcondor/manual/v7.6/2_10DAGMan_Applications.html`

2 What is Condor?

You can read about what Condor is and how it works in the PSI Condor wiki. Instead of running a job directly on a machine you submit a job to Condor and Condor determines where the job will run. A job is submitted with a job description file. Examples of such description files can be found at `https://wiki.esat.kuleuven.be/visics/condor_examples`.

3 Hello world

In the hello world folder you can find an example job that will print hello world. The script that will be executed is hello_world.sh and the job description file is hello_world.job. You can look inside the hello_world.job file to see what a job description file contains.

To submit the job open a terminal and go to the hello world folder. Then type:

```
condor_submit hello_world.job
```

This will put the hello world job in the Condor queue. You can look at the jobs in the Condor queue with the command:

```
condor_q
```

First the job will be listed as idle, which means it is waiting to be assigned to a machine. Once it is assigned to a machine it will run. The output of the job is written to hello_world.out. If everything went well this file should say 'hello world'.

4 Python

If you want to run a python (or similarly Matlab) script with Condor the job description file has to be changed somewhat. The executable is now python and the script that should be executed will be passed as an argument. You can find the python hello world in the hello python folder. If you submit hello_python.job, the result should be the same as the hello world example.

4.1 Using your own libraries

If you have installed some of your own libraries python will not be able to find them because the LD_LIBRARY_PATH environment variable is not set. To make sure python can find your libraries you should append the following to the script you execute:

```
import os, sys

if 'LD_LIBRARY_PATH' not in os.environ:
    os.environ['LD_LIBRARY_PATH'] = '/path/to/your/libfolder'
    try:
        os.system('/usr/bin/python ' + ' '.join(sys.argv))
        sys.exit(0)
    except Exception, exc:
        print 'Failed re-exec:', exc
        sys.exit(1)
```

This piece of code will check if the LD_LIBRARY_PATH environment variable has been defined. If not it will define it and re-execute the script.

5 Passing Arguments

It is possible to pass arguments when submitting a job with an append command. In the passing arguments folder you can find a job description file that contains a variable 'toprint'. It will execute a python script that prints whatever is in the toprint variable. You can submit this job with:

```
condor_submit -a "toprint = hello world" print.job
```

This will append the line 'toprint = hello world' to the job description file, so the script will print 'hello world'.

6 Parallel execution

The part where Condor really shines is its ability to process multiple jobs in parallel. Condor can execute up to 100 jobs in parallel and you can submit as many as you want to the queue. You can submit multiple job description files but there are smarter ways to submit many jobs at once.

6.1 Submitting independent jobs

Submitting many independent jobs can be done with only small modifications to the job description file. Independence in this context means that job A does not depend on the result of job B and vice versa. The final line of most job description files is the `Queue` command. Multiple jobs can be submitted by modifying this to `Queue x` where `x` is the number of jobs you want to submit.

The process id of each of the submitted jobs will be stored in the Process variable. This variable can be passed as an argument to your script and the script can then decide what to do based on the process ID. An example of a job description file that submits multiple jobs can be found in the multiple jobs folder. After you submit the job you will see that there are 10 jobs in the queue.

Every job will overwrite the the output file so it might be a good idea to define an output file for every job (based on the process ID). In this example each job will write to the jobs_output.txt file. If everything went well this file should contain a line for every job. However, you should be careful when writing to the same file from many different jobs because the jobs could collide when writing to that file simultaneously. A better way is to have a separate file for each process and then merge them when the jobs are all complete.

6.2 Submitting dependent jobs with DAGman

DAGman stands for Directed Acyclic Graph manager. DAGman runs on top of Condor and gives you the ability to define dependencies for your jobs. You can define parent-child relationships for jobs. DAGman will only submit the child if the parent has successfully terminated. DAGman adds more functionality to Condor, information about

DAGman can be found in the DAGman manual.

DAGman can be very useful for large jobs. Large jobs are not ideal for Condor because the job will block a machine for a long time. With DAGman you can cut your jobs in many small pieces. If your job is cut in small pieces Condor has more liberty to divide the job over different machines. When running a training algorithm with many iterations every iteration could, for example, be a single job.

An example of a DAGman job can be found in the dagman folder. This DAGman job will print the sentence 'this sentence is in the correct order.' and every job will print one word. The DAGman file can be found in dagman.dag and can be submitted as:

```
condor_submit_dag dagman.dag
```

When the job is finished the file dagman_output.txt should say 'this sentence is in the correct order.'.

As you can see in the dagman file, it is pretty verbose and if you have hundreds of jobs writing such a file will take a lot of effort. That is why it is often easier to programmatically write the file and then submit it. In the case of one job per iteration, for example, this should be pretty easy.

7 GPU

Some of the machines in the condor pool have a GPU while others do not. If your job requires a GPU you should specify this in the job description file as a requirement. These are some of the requirements you can set for the GPU's:

- CUDADeviceName
- CUDAOpenCLVersion
- CUDARuntimeVersion
- CUDACapability
- CUDAClockMhz
- CUDAGlobalMemoryMb
- CUDAComputeUnits
- CUDADriverVersion
- CUDACoresPerCU
- CUDAECCEnabled

You can print this information for a device with the command:

```
condor_status -long <device_name> | grep CUDA
```

In the gpu folder you can find a job that will print the CUDA information for the machine it's running on.