condorpy Documentation

Release 0.0.0

Scott Christensen

May 25, 2017

Contents

condorpy Python interface for high-throughput computing with HTCondor

Version 0.0.0

Author Scott Christensen

Team CI-Water

Homepage http://tethysplatform.org/condorpy

License BSD 2-Clause

Contents:

CHAPTER 1

User Manual

Creating Jobs

The CondorPy Job object represents an HTCondor job (which is sometimes called a cluster; see *Sub-Jobs*). A CondorPy Job automatically creates an HTCondor submit description file (or `job_file`). CondorPy seeks to simplify the process of creating HTCondor jobs as much as possible, but it is still helpful to understand how submit description files work. For a good overview see Submitting a Job in the HTCondor Users' Manual.

Creating a new CondorPy Job object is very easy and only requires a name:

```
from condorpy import Job

job = Job(name='my_job')
```

The job can then be configured by setting properties and attributes (see *Job Properties* and *Setting Job Attributes*). For convenience the Job constructor can also take a number of other arguments, which help to configure the job when it is created. For example:

```
from condorpy import Job, Templates

job = Job(name='my_job',
          attributes=Templates.base,
          num_jobs=5,
          host='example.com',
          username='root',
          private_key='~/.ssh/id_rsa',
          executable='my_script.py',
          arguments=('arg1', 'arg2')
          )
```

Here is a brief explanation of the arguments used in this example with links for additional details:

- `attributes`: A dictionary of job attributes. In this case a Template is used (see *Using Job Templates*).
- `num_jobs`: The number of sub-jobs that will be created as part of the job (see *Sub-Jobs*).

- `host`: The hostname or IP address of a remote scheduler where the job will be submitted (see *Remote Scheduling*).
- `username`: The username for the remote scheduler (see *Remote Scheduling*).
- `private_key`: The path the the private SSH key used to connect to the remote scheduler (see *Remote Scheduling*).
- `**kwargs`: A list of keyword arguments (in this example `executable` and `arguments`) that will be added to the attributes dictionary (see *Setting Job Attributes*).

For a full list of arguments that can be used in the Job constructor see the *API Reference*.

Job Properties

Jobs have the following properties (some of which may be set and others are read only):

- `name` (str): The name of the job. This is used to name the `job_file`, the `log_file`, and the `initial_directory`. The `job_name` job attribute in the `attributes` dictionary is also set by the `name` property.
- `attributes` (dict, read_only): A list of job attributes and their values. This property can be set in the Job constructor and can be modified by setting individual job attributes (see *Setting Job Attributes*).
- `num_jobs` (int): The number of sub-jobs that are part of the job. This can also be set when submitting the job (see *Sub-Jobs*).
- `cluster_id` (int, read_only): The id used to identify the job on the HTCondor scheduler.
- `status` (str, read_only): The status of the job.
- `statuses` (dict, read_only): A dictionary where the keys are all possible statuses and the values are the number of sub-jobs that have that status. The possible statuses are:
  - 'Unexpanded'
  - 'Idle'
  - 'Running'
  - 'Removed'
  - 'Completed'
  - 'Held'
  - 'Submission_err'
- `job_file` (str, read_only): The file path to the job file in the form [`initial_directory`]/[`name`].job
- `log_file` (str, read_only): The file path to the main log file and by default is [`initial_directory`]/logs/[`name`].log
- `initial_directory` (str, read_only): The home directory for the job. All input files are relative to this directory and all output files are written back to this directory (see *A Guide to File Paths*). By default the directory is created in the current working directory and is called [`name`]. This property comes from the `initialdir` job attribute.
- `remote_input_files` (list or tuple): A list or tuple of file paths to files that need to be transfered to a remote scheduler (see *Remote Scheduling*).

Setting Job Attributes

Job attributes are key-value pairs that get written to the job file (i.e. the HTCondor submit description file). These attributes ca

1. Using the `attributes` parameter in the Job constructor.
2. Using `**kwargs` in the Job constructor.
3. Using the `set` method of a job object.
4. Assigning values to attributes directly on a job object.

Valid job attributes are any of the commands that can be listed in the HTCondor submit description file. For a complete list and description of these commands see the HTCondor condor_submit documentation.

Using the `attributes` parameter in the Job constructor

The `attributes` parameter in the Job constructor accepts a dictionary, which becomes the attributes of the newly created job. This is often used to pass is a template that has pre-configured attributes, but it can be any dictionary object.

The following example uses a template to initialize a job with several job attributes using the `attributes` parameter of the Job constructor.

```
from condorpy import Job, Templates

job = Job(name='my_job', attributes=Templates.base)
```

This next example modifies a template to initialize a job with customized job attributes.

```
from condorpy import Job, Templates

my_attributes = Templates.base
my_attributes['executable'] = 'my_script.py'
my_attributes['arguments'] = ('arg1', 'arg2')

job = Job(name='my_job', attributes=my_attributes
```

Using `**kwargs` in the Job constructor

Additional job attributes can be set in the Job constructor by using keyword arguments (or kwargs).

In the following example `executable` and `arguments` are keyword arguments that get added as job attributes.

```
from condorpy import Job, Templates

job = Job(name='my_job', attributes=Templates.base, executable='my_script.py',␣
↪arguments=('arg1', 'arg2'))
```

Using the `set` method of a job object

Once an object has been instantiated from the Job class then attributes can be set using the `set` method.

In this example the `executable` and `arguments` attributes are set after the job has been created.

```
from condorpy import Job, Templates

job = Job(name='my_job', attributes=Templates.base)
job.set('executable', 'my_script.py')
job.set('arguments', ('arg1', 'arg2'))
```

Assigning values to attributes directly on a job object

For convenience job attributes can be assigned directly on the job object.

In the following example the `executable` and `arguments` attributes are set as attributes on the job object.

```
from condorpy import Job, Templates

job = Job(name='my_job', attributes=Templates.base)
job.executable = 'my_script.py'
job.arguments = ('arg1', 'arg2')
```

Sub-Jobs

It is often useful to have a single job execute multiple times with different arguments, or input data. This is what HTCondor calls a cluster of multiple jobs. In CondorPy it is said that the job has multiple sub-jobs. Creating multiple sub-jobs in CondorPy can be done in two ways: setting the `num_jobs` property of the job, or passing in a `queue` argument to the `submit` method, which also sets the `nubm_jobs` property.

```
# creating 100 sub-jobs by setting the num_jobs property
job.num_jobs = 100

# creating 100 sub-jobs by passing in a queue argument to the submit method
job.submit(queue=100)
```

Working Directory

Using Job Templates

Remote Scheduling

A Guide to File Paths

Workflows

CHAPTER 2

Overview of HTCondor

This is designed to be a basic, high-level overview of some of the key features and functionality of HTCondor. For a complete reference for HTCondor see the HTCondor User Manual.

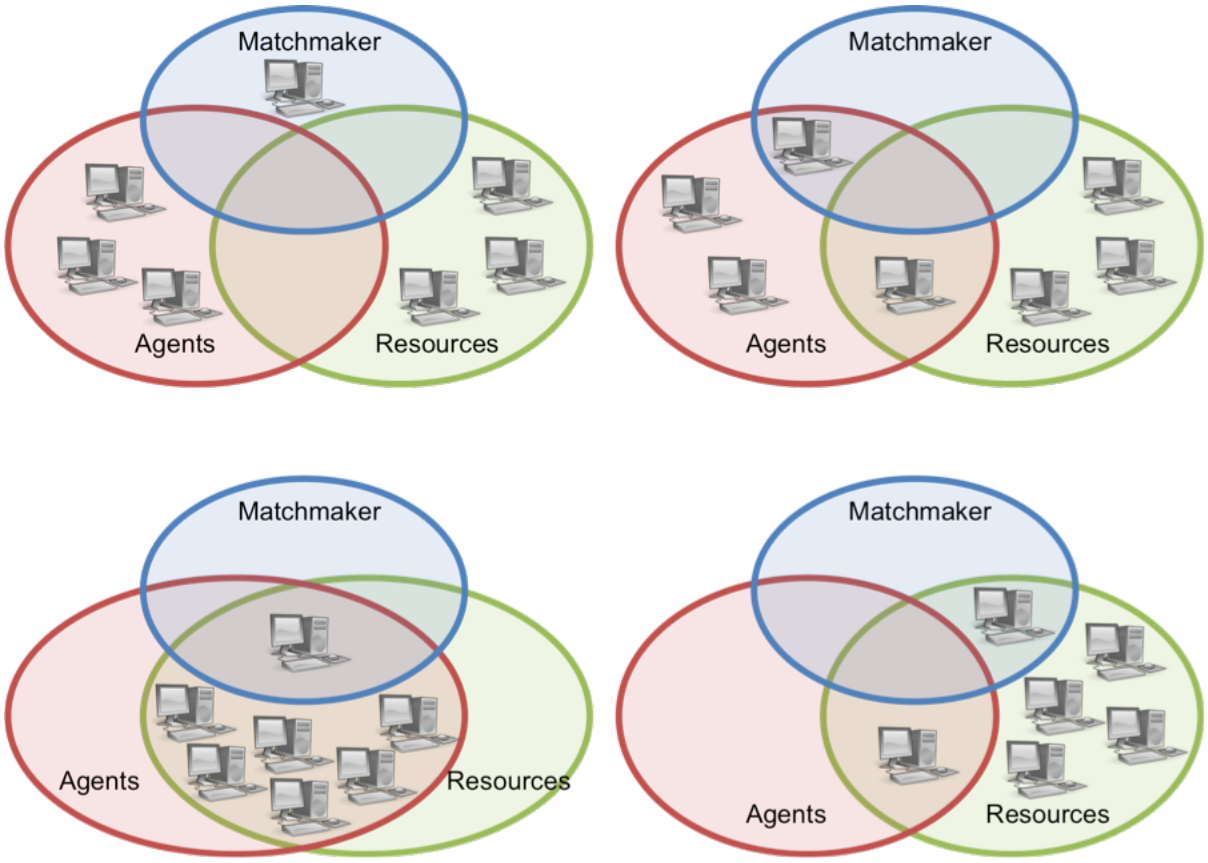
Warning: I am not an expert in HTCondor. I've worked with HTCondor for the last several years and have realized that there is a steep learning curve. This overview is designed to give a brief introduction to some of the main concepts of HTCondor as I understand them, for those that want to use HTCondor, but are not experts in the field of scientific computing.

What is HTCondor

HTCondor is a job scheduling and resource management software. HTCondor creates an HTC system by grouping, or "pooling", network-connected computing resources. A common example of an HTCondor pool is a set of lab or office computers that are all on the same network and configured (through HTCondor) to be part of the same computing system. Each computer in the pool is assigned one or more roles such as matchmaker, resource, or agent. Background processes, called daemons, which are specified in the computer's HTCondor configuration file determine the role(s) of a computer. Each pool has only one matchmaker, which serves as the central manager. All other computers in the pool are resources and/or agents and are configured to report to the central manager. A resource is also known as a worker, and is a computer designated to run jobs. And an agent, also known as a scheduler, is a computer designated to schedule jobs. Any computer in the pool (including the central manager) may function as both a resource and an agent. Figure 1 shows a possible HTCondor pool configuration.

Figure 1. Possible HTCondor pool configurations.

To submit a job to the pool a user must create a file to describe the job requirements, including the executable and any input or output files. This file is known as a problem solver. The user submits the problem solver to an agent, which advertises the requirements needed to run the job to the central manager. Similarly, each resource in the pool also advertises its availability, capabilities, and preferences to the central manager. This advertising, from both the agents and the resources, is done using a schema-free language called ClassAds. Periodically the central manager scans the ClassAds from resources and from agents and try to match jobs and resources that are compatible. When a match is made, the central manager notifies the agent and the resource of the match. It is then the agent's responsibility to



contact the resource and start the job. This interaction is handled by a process running on the agent, known as the shadow, which communicates with a process running on the resource, called the sandbox. The shadow provides all of the details required to execute a job, while the sandbox is responsible for creating a safe and isolated execution environment for the job.

Universes

HTCondor provides several runtime environments called universes. The two most common universes are the standard universe and the vanilla universe. The standard universe is only available on Unix machines. It enables remote system calls so the resource that is running the job can remotely make calls to the agent to open and read files, which means that job files need not be transferred, but can remain on the submitting machine. It also provides a mechanism called checkpointing. Checkpointing enables a job to save its state. Thus when a resource becomes unavailable in the middle of executing a job (because, for example, a user starts using the computer where the job is running), the job can start on a new resource from the most recent checkpoint. The vanilla universe is the default universe and is the most generic. It requires that the computing resources have a shared file system or that files be transferred from the submitting machine to the computing resource. The vanilla universe does not provide the checkpointing mechanism, and thus, if a job is interrupted mid-execution, it must be restarted on a new resource. Various other universes are available and are described in greater detail in the user manual

Checkpointing

Flocking

HTCondor also provides a mechanism, called flocking, for submitting jobs to more than one pool of resources. Flocking enables organizations to combine resources while maintaining control of their own pool. For a machine to be able to flock to a remote pool the remote scheduler must be identified in the configuration on that machine. Additionally, the remote scheduler must accept the machine to be flocked from in its own configuration. The submitting machine will first try to match jobs in its native pool, but when resources are not available then it will "flock" jobs to the remote pool. Generally, a remote pool will not have a shared file system, so jobs that are flocked must enable the file transfer mechanism.

DAGMan

HTCondor is ideal for embarrassingly parallel batch jobs, but it also provides a way of executing workflows using directional acyclic graphs (DAGs). A DAG specifies a series of jobs, referred to as nodes, that need to be run in a particular order and also defines the relationships between nodes using parent-child notation. This allows for the common situation where the output from a preliminary set of simulations is used as input for a subsequent set of simulations. An alternative scheduler called a DAG Manager (DAGMan) is used to orchestrate submitting jobs in the proper order to the normal scheduler. If a node in the DAG fails, the DAGMan generates a rescue DAG that keeps track of which nodes are completed and those that still need to run. A rescue DAG can be resubmitted, and it will continue the workflow from where it left off. This provides a robust mechanism for executing large workflows or a large number of jobs.

ClassAds

HPC vs. HTC

Under the Hood

CHAPTER 3

API Reference

condorpy package

Submodules

condorpy.job module

condorpy.node module

condorpy.templates module

condorpy.workflow module

Module contents

CHAPTER 4

Description

Condorpy is a wrapper for the command line interface (cli) of HTCondor and enables creating submitting and monitoring HTCondor jobs from Python. HTCondor must be installed to use condorpy.

CHAPTER 5

Installing

```
$ pip install condorpy
```

CHAPTER 6

Installing from Source

```
$ python setup.py install
```

CHAPTER 7

Getting Started

```
>>> from condorpy import Job, Templates
>>> job = Job('job_name', Templates.vanilla_transfer_files)
>>> job.executable = 'job_script'
>>> jobs.arguments = 'input_1 input_2'
>>> job.transfer_input_files = 'input_1 input_2'
>>> job.transfer_output_files = 'output'
>>> job.submit()
```

CHAPTER 8

TODO

CHAPTER 9

Acknowledgements

This material was developed as part of the CI-Water project which was supported by the National Science Foundation under Grant No. 1135482

CHAPTER 10

Indices and tables

- genindex
- modindex
- search