HTCondor and Python
Overview

› Python binding basics
› What's new since last year
› Some real life examples
>>> import htcondor
>>> import classad
>>> a = classad.ClassAd({"Hello": "World"})
>>> a
[ Hello = "World" ]
>>>
Python Basics

- `import htcondor; import classad`
- Use `dir()` to list object names in a module; use `help()` to get the per-method or class help.
- `print classad.version(), htcondor.version()`
- `htcondor.param['COLLECTOR_HOST']` to access parameter value of COLLECTOR_HOST.
In the beginning, there were ClassAds.

- ClassAds are the *lingua franca* of HTCondor-land.
- HTCondor CLI often converts ClassAds into human readable form or XML.
- The python bindings use the internal ClassAd objects throughout.
- We try to make it pleasant to convert between ClassAds and native Python objects.
ClassAds

```python
>>> import htcondor
>>> import classad
>>> a = classad.ClassAd({"Hello": "World"})
>>> a
[ Hello = "World" ]
>>> a["foo"] = "bar"
>>> a
[ foo = "bar"; Hello = "World" ]
>>> a["bar"] = [1,2,3]
>>> a
[ bar = { 1,2,3 }; foo = "bar"; Hello = "World" ]
>>> a["baz"] = classad.ExprTree("bar")
>>> a["baz"]
bar
>>> a.eval("baz")
[1, 2, 3]
```
The “htcondor” Python module allows you to interact with most HTCondor daemons.

There are three very important objects:
- Collector: read and write
- Schedd: submit and manipulate
- Negotiator: manage user and their priorities

And a few other helpers - enums, security manager, interaction with the config system, and sending daemons direct commands.
Collector Basics

- The Collector object allows one to locate daemons, query slot status, and advertise new ClassAds.
- The object takes the network location of the collector daemon for the constructor:
  ```python
coll = htcondor.Collector("red-condor.unl.edu")
  ```
- New: Can take a list of collectors for high-availability installations
Collector Basics

```python
>>> import htcondor
>>> coll = htcondor.Collector()
>>> ad = coll.locate(htcondor.DaemonTypes.Schedd, "submit-1.chtc.wisc.edu")
>>> ad["MyAddress"]
'\<128.104.100.43:9618?sock=2309_6dc1_2>\'
>>> ads = coll.locateAll(htcondor.DaemonTypes.Schedd)
>>> for ad in ads: print ad["Name"]
...
submit-1.chtc.wisc.edu
submit-2.chtc.wisc.edu
submit-3.chtc.wisc.edu
```
Collector Advanced

- For many queries, pulling all attributes from the collector is expensive.
- You can specify a *projection list* of attributes. HTCondor will return the minimum number of attributes containing the ones you specify.
- It will always pad in a few extra.
>>> import classad
>>> import htcondor
>>> coll = htcondor.Collector()
>>> ads = coll.query(htcondor.AdTypes.Startd, 'true',
                  ['MyAddress', 'Name', 'Cpus', 'Memory'])
>>> len(ads)
4994
>>> ads[0]
[ Name = "vmID.chtc.wisc.edu"; Cpus = 1; MyType = "Machine";
  Memory = 3737; MyAddress = "<192.168.78.147:36909?
  CCBID=128.105.244.14:9618#306906&PrivNet=vm.wisc.edu>";
  TargetType = "Job"; CurrentTime = time() ]
>>>
Schedd

Schedd methods

- submit: submit jobs
- query: get job ClassAds
- act: perform some action (hold, release, remove, suspend, continue) on one or more jobs
- edit: edit one or more job ClassAds
- reschedule: have schedd request a new negotiation cycle.
Submit ClassAds

- Normally we use submit files for jobs
- Python bindings use ClassAds
  - Translation from submit file to ClassAds
    - condor_submit -dump classads
    - Translations:
      - error / Err – should_transfer_files / ShouldTransferFiles
      - output / Out – transfer_input_files / TransferIn
      - executable / Cmd – transfer_output_files / TransferOut
  - Translation from macro to ClassAd language
    - This: error = “test.err.\$(Process)”
    - Becomes: Err = strcat(“test.err.”,ProcID)
>>> ad = classad.parse(open("test.submit.ad"))
# Submits two jobs in the cluster; edit test.submit.ad to preference.
>>> print schedd.submit(ad, 2)
110
>>> print schedd.act(htcondor.JobAction.Remove, ["111.0", "110.0"])
[
    TotalNotFound = 0;
    TotalPermissionDenied = 0;
    TotalAlreadyDone = 0;
    TotalJobAds = 2;
    TotalSuccess = 2;
    TotalChangedAds = 1;
    TotalBadStatus = 0;
    TotalError = 0
]
Daemon Commands

› Administrator can send arbitrary commands to HTCondor daemon via python
   Uses same protocol as condor_off and condor_on
   Similar to unix signals, the command is sent, no indication that command did anything

› A new developer should keep to Reconfig, Restart, and DaemonsOff
   Some commands will take an extra argument – such as subsystem for “DaemonOff”.

› htcondor.send_command(ad, htcondor.DaemonCommands.Restart)
Daemon Commands

› We hope this will improve the “scriptability” of a HTCondor pool
  - For example, one could implement a rolling restart cron job that ensures no more than 10% of nodes are draining at once.
To invalidate an existing in-process security session:

- `htcondor.SecMan().invalidateAllSessions()`

To access the param subsystem:

- `htcondor.param`
- *Treat like a python dictionary.*

To reload the client configuration from disk:

- `htcondor.reload_config()`
What's new since last year

› Negotiator class to manage users and their priorities
› Python 3 support
› Refresh GSI proxy
› Collector class can be initialized with a list of collectors
› Can ping HTCondor daemons
What's new (cont.)

› HTCondor bindings can accept python dictionaries as well as ClassAds
› Improved ClassAd expression building
› Stream parser of ClassAds
› Support to read event logs
› Support for schedd transactions
Ping Daemons

- `htcondor.SecMan().ping(schedd_ad)`
Accept python dictionaries

Example before:

```python
import htcondor
import classad
htcondor.Schedd().submit(classad.ClassAd({"Cmd": "/bin/echo"}))
```

Example after:

```python
import htcondor
htcondor.Schedd().submit({"Cmd": "/bin/echo"})
```
ClassAd expressions

› Added quote() and unquote() to convert between python and ClassAd string literals
  ▶ classad.ExprTree("foo =?= %s" %
     classad.quote("bar"))

› ClassAd expression building greatly improved
  ▶ foo = "bar"
  ▶ expr = classad.Attribute("MyType") ==
     classad.Literal(foo)
ClassAd Stream Parser

```python
>>> import classad
>>> import os
>>> fd = os.popen("condor_history -l -match 4")
>>> ads = classad.parseOldAds(fd)
>>> print [ad["ClusterId"] for ad in ads]
[23389L, 23388L, 23386L, 23387L]
```
Read Event Logs

› `htcondor.read_events(open("LogFile"))`

› Returns a python iterator of ClassAds
Schedd transactions

```python
with schedd.transaction() as txn:
    schedd.submit(...)
    schedd.edit(...)
```

- Variable `txn` is destroyed at end of block
- If exception is thrown transaction is aborted
- Otherwise, it is committed upon destruction
Real World Examples

› Counting Job ads in Running State
› Invalidating stale startd ads
#!/usr/bin/python
import htcondor

collector = htcondor.Collector()

jn = 0
for schedd_ad in collector.locateAll(htcondor.DaemonTypes.Schedd):
    schedd = htcondor.Schedd(schedd_ad)
    ads = schedd.query('JobStatus == 2', ['ClusterId'])
    jn += len(ads)
print jn
Invalidate Startds

#!/usr/bin/python
# Flush stale machine classads

import optparse
import htcondor
coll = htcondor.Collector(opts.pool)
ads = coll.query(htcondor.AdTypes.Startd, opts.constraint, ['MyAddress', 'Name'])
for a in ads: print a['Name']
if (opts.doit):
    coll.advertise(ads, "INVALIDATE_STARTD_ADS")
Questions