HTCondor Python Tutorial

Brian Bockelman
Welcome, Python!

• HTCondor has long provided two APIs into its ecosystem:
  • Traditional POSIX API: Input via stdin / argv, Output via stdout, error via exit codes.
  • SOAP: RPC-oriented API. Language-agnostic.
• Both have drawbacks. Fork overhead, parsing overhead, no reuse of security sessions.
• Python is a special case - widely utilized by projects in which build on top of HTCondor.
  • ... and Boost.Python makes it straightforward to write/maintain bindings...
  • HTCondor includes python bindings for most client-side activities since 7.9.4.
Audience!

- The audience for the python bindings are integrators/developers - we consciously expose lower-level interfaces than the CLI.
- If you want a more straightforward way to interact with HTCondor via python, this tutorial is for you!
- Some are decently refined; some are pretty raw wrappers around C++.
- These are one of the most powerful ways of programmatically interacting with the system.
- Not the simplest. (Yet?)
- I assume basic python and intermediate/advanced HTCondor knowledge.
Tutorial Time

• In this tutorial, I plan on covering the basics of using the python bindings.

• You’ll need your Linux-based laptop out with a fresh install of HTCondor >= 7.9.5.

  • Startup a personal HTCondor instance. Verify you can run basic commands (condor_submit, _status, _q).

  • For the most part, this will be “follow along Brian’s terminal”, but slides are here for later students.

  • (And in case the network connection explodes.)
Login yourself

• Hostname: ec2-54-224-238-91.compute-1.amazonaws.com
• User: demo
• Pass: theHTissilent
Hello, (HTCondor) 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.
- Condor CLI often converts ClassAds into human readable form or XML.
- The python bindings use the internal ClassAd objects throughout.
- ClassAds may look like bastardized JSON, but there are important evaluation semantics we can take care of.
- We try to make it pleasant to convert between ClassAds and native Python objects.
ClassAds

Sub-ClassAds are supported too!

```python
>>> classad.ClassAd({"foo": {"bar": True}})
[ foo = [ bar = 1 ] ]
```
HTCondor Module

- The “htcondor” Python module allows you to interact with most HTCondor daemons.
- There are two very important objects:
  - Collector: read and write
  - Schedd: submit and manipulate
- 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:
  
  • coll = htcondor.Collector("red-condor.unl.edu")
Collector Basics

Last login: Sat Apr 27 11:26:34 on ttys004
Brians-MacBook-Air:~ bbockelm$ python
Python 2.7.2 (default, Jun 20 2012, 16:23:33)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import htcondor
>>> ad = htcondor.Collector("red-condor.unl.edu")
>>> ad["MyAddress"]
'129.93.239.129:41562'
>>> for ad in ad.list(): print ad["Name"]
red-gw1.unl.edu
red-gw2.unl.edu
red.unl.edu
flocking@t3.unl.edu
t3.unl.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.
Collector - Advanced

```
Last login: Mon Apr 29 08:47:21 on ttys004
Brians-MacBook-Air:~ bbockelm$ python
Python 2.7.2 (default, Jun 20 2012, 16:23:33)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import classad
>>> import htcondor
>>> coll = htcondor.Collector("red-condor.unl.edu")
>>> ads = coll.query(htcondor.AdTypes.Startd, 'true', ['MyAddress', 'Name', 'Cpus', 'Memory'])
>>> len(ads)
4649
>>> ads[0]
{'CurrentTime': 1332567530, 'Memory': 119, 'MyAddress': '<172.16.1.70:38406>', 'Cpus': 0, 'Name': 'slot1@node070.red.hcc.unl.edu', 'MyType': 'Machine', 'TargetType': 'Job'}
```
Schedd Basics

```
Last login: Mon Apr 29 14:25:48 on ttys004
Python 2.7.2 (default, Jun 20 2012, 16:23:33)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import htcondor
>>> coll = htcondor.Collector("red-condor.unl.edu")
>>> schedd_ad = colllocate(htcondor.DaemonTypes.Schedd, "red.unl.edu")
>>> schedd = htcondor.Schedd(schedd_ad)
>>> jobs = schedd.query()
>>> print jobs[0],
[
    {'CurrentTime': time(),
     'BufferSize': 524288,
     'JobNotification': 0,
     'CoreSize': 32768,
     'Err': '/var/lib/globus/job_home/uscmsPool2295/.globus/job/red/1629003077231732616.19054336811412418718/stcerr',
     'CumulativeSlotTime': 0,
     'CoreSize': -1,
     'NiceUser': false,
     'x09UserProxyExpiration': 1367424189}
```

```
Job", CondorVersion = "$CondorVersion: 7.9.5 Apr 04 2013 BuildID: 114739 $";
obRunCount = 1; StreamErr = false; DiskUsage_RAW = 1; PeriodicHold = false; Proc
Id = 0; User = "demo@10-62-61-234.ec2.internal"; TransferQueued = false; Last
JobStatus = 1; Arguments = "-c 'echo Hello world & sleep 1m' "; Out = "test.out";
JobCurrentStartDate = 1367273629; JobStatus = 2; PeriodicRelease = false; Auto
ClusterAttrs = "JobUniverse,LastCheckpointPlatform,NumCkpts,RemoteGroup,Submitte
rGroup,SubmitterUserPri,DiskUsage,ImageSize,RequestDisk,RequestMemory,Requirem
ents,NiceUser,ConcurrencyLimits"; RequestMemory = ifthenelse(MemoryUsage isnt and
defined,MemoryUsage, (ImageSize + 1023) / 1024); Args = ""; MaxHosts = 1; TotalS
suspensions = 0; CommittedSlotTime = 0; StartedPrincipal = "unauthenticated@unnam
ed/10.62.61.234"; CondorPlatform = "$CondorPlatform: x86_64_Hat6 $"; AutoClus
terId = 7; ShouldTransferFiles = "YES"; ExitStatus = 0; NumShadowStarts = 1; Mac
hineAttrCpus0 = 1; QDate = 1367273629; EnteredCurrentStatus = 1367273629]]
```
Submit ClassAds

- We normally submit using the submit file format, not using ClassAds.
- Switching to ClassAds for submission requires a rewiring a few neurons.
- Realizing the differences between the macro and ClassAd language costs a few more neurons.
- A few submit file / ClassAds translations:
  - error / Err
  - output / Out
  - executable / Cmd
  - should_transfer_files / ShouldTransferFiles
  - transfer_input_files / TransferIn
  - transfer_output_files / TransferOut
- The second argument to Schedd.submit determines how many processes to submit.
- From macros to ClassAds:
  - Instead of: error = “test.err.$(Process)”
  - Write: Err = strcat(“test.err”, ProclID)
Schedd Advanced

• A few useful methods:
  • **act**: Perform some action on one or more jobs (hold, release, remove, removeX, suspend, continue).
  • **edit**: Edit one or more job ClassAds
  • **reschedule**: Have Schedd request a new negotiation cycle.
Schedd Advanced

```python
[demo@ip-10-62-61-234 ~]$ python
Python 2.6.6 (r266:84292, Dec 7 2011, 20:48:22)
[GCC 4.4.6 20110731 (Red Hat 4.4.6-3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import htcondor
>>> schedd = htcondor.Schedd()
>>> jobs = schedd.query('true', ['ClusterID', 'foo'])
>>> jobs
[[ MyType = "Job"; TargetType = "Machine"; ServerTime = 1367284751; ClusterID = 7; CurrentTime = time() ]]
>>> schedd.edit('ClusterID == 7', "foo", "bar")
>>> schedd.query('true', ['ClusterID', 'foo'])
[[ MyType = "Job"; foo = "bar"; TargetType = "Machine"; ServerTime = 1367284782; ClusterID == 7; CurrentTime = time() ]]
>>> schedd.edit('ClusterID == 7', "foo", "42")
>>> schedd.query('true', ['ClusterID', 'foo'])
[[ MyType = "Job"; foo = "42"; TargetType = "Machine"; ServerTime = 1367284792; ClusterID == 7; CurrentTime = time() ]]
>>> schedd.act(htcondor.JobAction.Hold, ['7.0'])
[ TotalNotFound = 0; TotalPermissionDenied = 0; TotalAlreadyDone = 1; TotalJobAdd = 1; TotalSuccess = 0; TotalChangedAds = 0; TotalBadStatus = 0; TotalError = 0 ]
```
Schedd Advanced - File Transfer

```python
>>> schedd.query('ClusterID == 5', "["ClusterID", "ProcID"]")
[[ MyType = "Job"; TargetType = "Machine"; ServerTime = 1367275365; ClusterID = 5; ProcID = 0; CurrentTime = time() ]]
>>> schedd.query('ClusterID == 5', "["ClusterID", "ProcID", "JobStatus"]")
[[ MyType = "Job"; JobStatus = 5; TargetType = "Machine"; ServerTime = 1367275393; ClusterID = 5; ProcID = 0; CurrentTime = time() ]]
>>> schedd.spool(ad_results)
>>> schedd.query('ClusterID == 5', "["ClusterID", "ProcID", "JobStatus"]")
[[ MyType = "Job"; JobStatus = 5; TargetType = "Machine"; ServerTime = 1367275411; ClusterID = 5; ProcID = 0; CurrentTime = time() ]]
>>> schedd.query('ClusterID == 5', "["ClusterID", "ProcID", "JobStatus"]")
[[ MyType = "Job"; JobStatus = 4; TargetType = "Machine"; ServerTime = 1367275471; ClusterID = 5; ProcID = 0; CurrentTime = time() ]]
>>> schedd.retrieve('ClusterID == 5')
>>> schedd.query('ClusterID == 5', "["ClusterID", "ProcID", "JobStatus"]")
[[ MyType = "Job"; JobStatus = 4; TargetType = "Machine"; ServerTime = 1367275504; ClusterID = 5; ProcID = 0; CurrentTime = time() ]]
>>> schedd.act(htcondor.JobAction.Remove, ["5.0")
[ TotalNotFound = 0; TotalPermissionDenied = 0; TotalAlreadyDone = 0; TotalJobAds = 1; TotalSuccess = 1; TotalChangedAds = 1; TotalBadStatus = 0; TotalError = 0 ]
```
Daemon Commands

• An administrator can send commands to arbitrary HTCondor daemons via python.
  • Uses the same internal protocol as CLI such as condor_off and condor_on.
  • A blessing and a curse: HTCondor doesn’t document what the protocol commands do. They are a bit similar to Unix signals in that you receive no indication the command did anything.

• Do you know the difference between DaemonOff, DaemonOffFast, DaemonOffPeaceful, DaemonsOff, DaemonsOffFast, DaemonsOffPeaceful, OffFast, OffForce, OffGraceful, and OffPeaceful?

• A new developer best keep to Reconfig, Restart, and DaemonsOff. Send the command to the master.

• Some commands will take an extra argument - such as the subsystem to restart for “DaemonOff”.
Daemon Commands

```
[demo@ip-10-62-61-234 ~]$ python
Python 2.6.6 (r266:84292, Dec 7 2011, 20:48:22)
[GCC 4.4.6 20110731 (Red Hat 4.4.6-3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import htcondor
>>> coll = htcondor.Collector()
>>> ad = coll.locate(htcondor.DaemonTypes.Master, 'ip-10-62-61-234.ec2.internal')
>>> print ad['MyAddress'], ad['Name'],<10.62.61.234:34494> ip-10-62-61-234.ec2.internal
>>> htcondor.send_command(ad, htcondor.DaemonCommands.Restart)

[demo@ip-10-62-61-234 ~]$ tail -f /var/log/condor/MasterLog
04/29/13 23:00:59 /etc/condor/condor_config.local
04/29/13 23:00:59 DaemonCore: command socket at <10.62.61.234:43275>
04/29/13 23:00:59 DaemonCore: private command socket at <10.62.61.234:43275>
04/29/13 23:00:59 Setting maximum accepts per cycle 8.
04/29/13 23:01:00 started DaemonCore process "/usr/sbin/condor_collector", pid and pgroup = 20604
04/29/13 23:01:00 Waiting for /var/log/condor/.collector_address to appear.
04/29/13 23:01:01 Found /var/log/condor/.collector_address.
04/29/13 23:01:01 started DaemonCore process "/usr/sbin/condor_negotiator", pid and pgroup = 20605
04/29/13 23:01:01 started DaemonCore process "/usr/sbin/condor_schedd", pid and
```
Daemon Commands

- I hope this will really 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.
Etc

- 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()`
Python Bindings
Futures

• Python bindings will continue to receive periodic updates to keep parity with client-side tools. Current plans for 8.0:
  • Improve Schedd.edit method.
  • Release bindings for Mac OS X.
  • Better implement keyword parameters throughout.
  • Config errors should not exit the interpreter.

• Wishlist:
  • Add bindings for condor_tail.
  • Add bindings for condor_ping.
  • Expose more advanced ClassAd functionality (matching).
  • Cleanup the send_command function.
  • DaemonCore?

• I’m looking to broaden the set of maintainers. If you want seriously better bindings, plan to contribute!
  • In particular, I have no knowledge of Windows development! I believe we are a few small patches away from enabling Python bindings for Windows.