Manuale HTCondor

Indice

Login

Per effettuare il login è necessario usare il protocollo ssh mediante il seguente comando:

```
> ssh -i <path_private_key> <username>@frontend.recas.ba.infn.it
```

Il login avviene sulla home dell'utente:

```
> pwd
/lustrehome/<username>
```

Top

Sottomettere un job in HTCondor

Supponiamo di voler eseguire il comando echo per stampare a schermo la stringa “Hello World!”.
Per sottomettere un job in HTCondor è necessario un submit descriprion file, un file di testo che descrive il job da sottomettere (percorso delle’eseguibile, requisiti, ecc...).
Una più dettagliata spiegazione relativa al submit description file e ai suoi comandi è riportata in seguito.

Submit description file

```
# This is a comment in the submit file
# file name : submit_echo
# universe describes an execution envirorment. Set universe to vanilla
universe = vanilla
# Path of the executable (it can be a system command, your own application, a
script, etc...)
executable = /bin/echo
# The argument to pass to the executable
arguments = "Hello World!"
# The output of the remote machine running the job will be printed on
echo.out
output = echo.out
error = echo.error
log = echo.log
request_cpus = 1
rank = Memory
queue
```

`Nota:` se non vengono specificati `request_cpus`, `request_disk`, `request_memory` nel submit file, il job verrà eseguito su una quantità di default di cpu(1), memoria e spazio sul disco.

Per sottomettere il job usiamo `condor_submit` passando come argomento il nome del submit file:

```
> condor_submit submit_echo -name ettore
```

Top

Perchè l'opzione -name<schedd>?

Al fine di sottomettere i comandi relativi ai job è necessario specificare uno schedd, ergo l'opzione `-name<schedd>` nella sottomissione del comando poco più in alto.

Tutti i comandi descritti in questo documento necessitano dell'opzione `-name ettore` per funzionare, fatta eccezione per `condor_status`.

Top

Sottomettere un job

`condor_submit` è il comando per la sottomissione di job per l'esecuzione attraverso HTCondor.

```
> condor_submit <path_submit_file>
```

```
> condor_submit <path_submit_file> -interactive
```

Indica che l'utente vuole eseguire una shell interattiva sulla macchina remota che esegue il job.

Top

Di seguito si riportano due esempi di submit file:

Il seguente submit file manda in esecuzione l'applicazione mathematica richiedendo al batch system di allocare per la sua esecuzione 1GB di RAM:

```
executable      = mathematica
universe        = vanilla
input           = test.data
output          = loop.out
error           = loop.error
log             = loop.log
request_memory  = 1 GB
initialdir      = run_1
queue
```

Il seguente submit file manda in esecuzione l'applicazione mathematica richiedendo al batch system di allocare per la sua esecuzione 24GB di RAM e 8 cpu:

```
executable      = mathematica
universe        = vanilla
input           = test.data
output          = loop.out
error           = loop.error
log             = loop.log
request_memory  = 24 GB
request_cpus    = 8
initialdir      = run_1
queue
```

HTCondor monitora l'uso delle risorse da parte dei batch job in esecuzione controllando che le risorse in uso non superino quelle richieste nel submit file. In tal caso il job viene killato.

Richiedere troppe risorse, in particolare le cpu, potrebbe allungare decisamente i tempi di attesa del job prima di vederlo in esecuzione.

Controllare lo stato dei job sottomessi

Ottenere l'ID e altre informazioni relative ad un job

`> condor_q`

Mostra informazioni di tutti i job in coda. Per ogni job è mostrato:

- id: id del job
- status: I – idle (in attesa di una macchina che lo esegua), R – in esecuzione, H – in attesa, S – sopseso, C – completato, X – rimosso, < – trasferimento input, > – trasferimento output
- altre informazioni (come tempo di esecuzione (RUNTIME), dimensione, priorità (PRI), nome dell'eseguibile (CMD))

Top

Visualizzare i job in esecuzione

`> condor_q -run`

Top

Visualizzare solo i propri job

`> condor_q -constraint 'OWNER == "<username>"'`

Top

Visualizzare i job in attesa

`> condor_q -hold`

Top

Visualizzare uno specifico job

`> condor_q <job_id>`

Top

Perchè il job non completa la sua esecuzione?

`> condor_q -better-analyze <job_id>`

Effettua una dettagliata analisi di matchmaking per determinare quanti slot sono disponibili per eseguire il job

Top

Controllare l'output di un job in esecuzione

`> condor_tail <job_id>`

Mostra le ultime righe di `stdout` di un job in esecuzione

Top

Gestire un job

Rimuovere un job dalla coda

Un job può essere rimosso dalla coda in qualsiasi momento usando il comando `condor_rm`.

```
> condor_rm <job_id>
```

Top

Porre un job in attesa

Un job può essere posto in attesa mediante il comando `condor_hold`.
Quando un job è posto in attesa, non verrà schedulato per l'esecuzione finchè non viene rilasciato. Se il job è in esecuzione mentre `condor_hold` viene invocato, il job verrà rimosso dalla macchina sulla quale il job è in esecuzione.

```
> condor_hold <job_id>
```

Top

Rilasciare un job

A job può essere rilasciato mediante il comando `condor_release`.
Quando un job posto in attesa viene rilascisto, tornerà allo stato idle, e varrà schedulato all'esecuzione appena possibile. Solo i job posti in attesa possono essere rilasciati.

```
> condor_release <job_id>
```

Top

Cambiare la priorità dei job

La priorità dei job può essere cambiata mediante il comando `condor_prio`.
La priorità di un job può essere un qualsiasi intero, dove numeri più grandi corrispondono a una priorità più alta; + *value* aumenta la priorità del job della quantità data da *value*. - *value* riduce la priorità del job della quantità data da *value*.

```
> condor_prio -p <+|-value> <job_id>
```

Top

Controllare lo stato del pool

`condor_status` è un utile strumento che può essere usato per mostrare e monitorare il pool in HTCondor.

`Nota:` `condor_status` non necessita dell'opzione `-name<schedd>` per funzionare.

`> condor_status` **`-available`**
Mostra gli slot disponibili

`> condor_status` **`-available -autoformat`** `Name Memory Cpus Disk`
Mostra nome, memoria, cpu, spazio sul disco degli slot disponibili

`> condor_status` **`-run`**
Mostra gli slot che eseguono job nel momento corrente

Top

Visualizzare gli slot che eseguono i propri job

```
> condor_status -constraint 'RemoteUser == "<username>@ReCaSCluster"'
```

Top

Accedere a un job running

È possibile accedere tramite ssh a un proprio job running attraverso il comando

```
> condor_ssh_to_job -name <schedd> <job_id>
```

in cui bisogna sostituire a `<schedd>` il nome dello schedd a cui è stato sottomesso il job (e.g. ettore), e a `<job_id>` l'id del job.

Viene quindi avviata una connessione remota con lo stesso user del job e nella cartella da cui il comando è stato lanciato, e viene fornito un PID associato al job. Per uscire dalla sessione, digitare `logout`.
Altre opzioni possono essere visualizzate con

```
> condor_ssh_to_job -help
```

Ulteriori informazioni e alcuni esempi di utilizzo possono essere trovati a questo link.

Top

Submit description file

Un submit description file è un file di testo che contiene tutto ciò che HTCondor ha bisogno di sapere sul job da essere sottomesso come il nome dell'eseguibile, la working directory iniziale, argomenti da passare all'eseguibile, ecc...

Comandi obbligatori

`universe = vanilla`

`universe` rappresenta un ambiente di esecuzione. Impostare questo comando a `vanilla`.

`executable = <path_name>`

Percorso dell'eseguibile

`queue`

Questo comando invia il job alla coda perciò dev'essere l'ultimo comando nel submit file

Top

Comandi utili

`input = <path_name>`

Questo file deve contenere tutti gli input da tastiera che il programma richiede (questo file è praticamente `stdin`). Il carattere di fine linea equivale a premere `INVIO` sulla tastiera.

`output = <path_name>`

Questo file contiene l'output che il programa scrive sulla macchina remota (questo file è praticamente `stdout`).

`error = <path_name>`

Questo file contiene i messaggi di errore che il programma normalmente stamperebbe a schermo (questo file è praticamente `stderr`).

`log = <path_name>`

Questo file contiene informazioni relative a ciò che succede mentre il job è in esecuzione. Se questo file è specificato nel submit file, HTCondor aggiungerà una log entry per svariati eventi, per esempio quando il job viene eseguito, viene spostato su un'altra macchina, il job viene completato

`arguments = <arguments list>`

Lista di argomenti da fornire all'eseguibile come parte della linea di comando.
In HTCondor ci sono due possibili formati per specificare gli argomenti, conosciuti come vecchia sintassi e nuova sintassi. Per dettagli ed esempi clicca qui.
Digitando il seguente comando:

```
> condor_q <job_id> -long -autoformat args arguments
```

Se è stata usata la vecchia sintassi verra mostrato a schermo l'attributo `args`, se è stata usata la nuova sintassi verrà mostrato a schermo l'attributo `arguments`

`priority = <+|- value>`

Assegna una priorità al job. Maggiore il valore, più alta la priorità

Top

Aggiungere requisiti alle macchine che eseguiranno il job

`request_cpus = <num_cpus>`

Numero di CPU richiesta per il job. Se non specificato, il numero di CPU per il job è 1.

`request_disk = <kilobytes>`

Quantità di spazio sul disco in KB richiesta per il job. Se non specificato, al job verrà assegnata una quantità di default di spazio sul disco.

`request_memory = <megabytes>`

Quantità di memoria in MB richiesta per il job. Se non specificato, al job verrà assegnata una quantità di default di memoria.

`requirements = <ClassAd expression>`

Esegui il job sulle macchine che soddisfano l'espressione ClassAd

`rank = <ClassAd expression>`

Il comando `rank` assegna un ordine alle macchine in base all'espressione ClassAd ed esegue il job sulla macchina che si posiziona più in alto nell'ordine. Il comando `rank` e i comandi di requisiti descritti sopra possono coesistere nello stesso file: mentre i comandi di requisiti escludono una macchina che non soddisfano i criteri dell'utente dall'eseguire il job, il comando `rank` eseguirà il job sulla macchina migliore, sempre secondo i criteri dell'utente, disponibile nel momento corrente.

Top

A proposito di `requirements` e `rank`

Sia `requirements` che `rank` devono essere validi espressioni ClassAd. In una espressione ClassAd i nomi degli attributi sono case insensitive mentre i valori sono case sensitive. Perciò le seguenti espressioni ClassAd sono valide:

```
Requirements = OpSys == "LINUX" && Arch == "INTEL"
requirements = opsys == "LINUX" && arch == "INTEL"
```

mentre la seguente no:

```
requirements = opsys == "linux" && arch == "intel"
```

Top

Altri job di esempio

1. Sottomissione multipla

Il submit file di seguito esegue 5 istanze del programma `sleep`; per ogni esecuzione l'id del processo relativo alla singola esecuzione è passato come argomento:

Submit description file

```
universe       = vanilla
executable     = /bin/sleep
arguments      = $(Process)
log            = sleep.log
queue 5
```

Nota: Il job id in HTCondor è definito come `Cluster.Process`. Se vengono sottomessi più job dallo stesso submit file, ogni job avrà lo stesso Cluster ID

Top

2. Altro sulla sottomissione multipla

Il submit file di seguito esegue 3 istanze del programma `echo` e per ogni esecuzione viene stampato su `stdout` una stringa diversa

Submit description file

```
universe      = vanilla
executable    = /bin/echo
output        = job_$(Cluster).$(Process).out
error         = job_$(Cluster).$(Process).err
log           = job_$(Cluster).log

arguments     = "String 1"
queue

arguments     = "String 2"
queue

arguments     = "String 3"
queue
```

Per la x esecuzione:

- `stdout` verrà inviato a `job_idcluster.x.out`
- `stderr` verrà inviato a `job_idcluster.x.err`

Top

3. Lavorare con i file

Poichè il pool utilizza un file system condiviso non è necessario specificare nel submit file il trasferimento file da e verso la macchina remota che esegue il job. È sufficiente assicurarsi che i path dei file (assoluti o relativi) che il programma necessita siano validi.
Supponiamo di voler eseguire un semplice bash script che stampa a schermo il contenuto di un file di testo:

```
#!/bin/sh
while read line; do
    echo "$line"
done < file.txt
```

Submit description file

```
universe     = vanilla
executable   = script.sh
log          = script.log
output       = script.out
queue
```

Top

4. Sottomissione multipla e file

Il submit file di seguito esegue 2 istanze di un programma chiamato `myprogram` che per funzionare necessita di input da `stdin`, un file di testo e di un argomento; una volta che la sua esecuzione è completa produce un file di output chiamato `out.txt`
Utilizzando il comando `initialdir` è possible specificare una differente directory di partenza per ogni job evitando in questo esempio la sovrascrittura del file di output `out.txt`

Submit description file

```
universe     = vanilla
executable   = myprogram
log          = myprogram.log
input        = myprogram.in
output       = myprogram.out
error        = myprogram.err
initialdir   = run_1
arguments    = a
queue
initialdir   = run_2
arguments    = w
queue
```

Per la x esecuzione:

- `stdout(stderr)` verrà inviato a `run_x/myprogram.out(run_x/myprogram.err)`
- `stdin` sarà letto da `run_x/myprogram.in`

Top