HTCondor IAC User's Manual

Ángel de Vicente*

February 17, 2023

Contents

1 Users' documentation for the IAC HTCondor Pool

1.1 An Overview of the HTCondor System

1.1.1 On High-Throughput Computing

For many scientists, the quality of their research is heavily dependent on computing throughput. It is not uncommon to find problems that require weeks or months of computation to solve. Scientists involved in this type of research need a computing environment that delivers large amounts of computational power over a long period of time. Such an environment is called a High-Throughput Computing (HTC) environment. In contrast, High-Performance Computing (HPC) environments deliver a tremendous amount of power over a short period of time. HPC environments are often measured in terms of FLoating point OPerations per Second (FLOPS). Many scientists today do not care about FLOPS; their problems are on a much larger scale. These people are concerned with floating point operations per month or per year. They are interested in how many jobs they can complete over a long period of time.

A key to high throughput is the efficient use of available resources. Years ago, the scientific community relied on large mainframe computers to do computational work. A large number of individuals and groups would have to pool their financial resources to afford such a computer. It was not uncommon to find just one such machine at even the largest research institutions. Scientists would wait their turn for mainframe time, and they would be allocated a specific amount of time. Scientists limited the size and scope of their problems to ensure completion. While this environment was inconvenient for the users, it was very efficient, because the mainframe was busy nearly all the time.

As computers became smaller, faster and less expensive, scientists moved away from mainframes and purchased personal computers or workstations. An individual or a small group could afford a computing resource that was available whenever they wanted it. The resource might be slower than the mainframe, but it provided exclusive access. Recently, instead of one large computer for an institution, there are many workstations. Each workstation is owned by its user. This is distributed ownership. While distributed ownership is more convenient for the users, it is also less efficient.

*angel.de.vicente@iac.es

Machines sit idle for long periods of time, often while their users are busy doing other things. **Condor takes this wasted computation time and puts it to good use.** The situation today matches that of yesterday, with the addition of clusters in the list of resources. These machines are often dedicated to tasks. Condor manages a cluster's effort efficiently, as well as handling other resources.

To achieve the highest throughput, Condor provides two important functions. First, it makes available resources more efficient by putting idle machines to work. Second, it expands the resources available to users, by functioning well in an environment of distributed ownership.

1.1.2 Why use Condor?

Condor takes advantage of computing resources that would otherwise be wasted and puts them to good use. Condor streamlines the scientist's tasks by allowing the submission of many jobs at the same time. In this way, tremendous amounts of computation can be done with very little intervention from the user. Moreover, Condor allows users to take advantage of idle machines that they would not otherwise have access to.

Condor provides other important features to its users. Source code does not have to be modified in any way to take advantage of these benefits. Code that can be re-linked with the Condor libraries gains two further abilities: the jobs can produce checkpoints and they can perform remote system calls.

A checkpoint is the complete set of information that comprises a program's state. Given a checkpoint, a program can use the checkpoint to resume execution. For long-running computations, the ability to produce and use checkpoints can save days, or even weeks of accumulated computation time. If a machine crashes, or must be rebooted for an administrative task, a checkpoint preserves computation already completed. Condor makes checkpoints of jobs, doing so periodically, or when the machine on which a job is executing will shortly become unavailable. In this way, the job can be continued on another machine (of the same platform); this is known as process migration.

A user submits a job to Condor. The job is executed on a remote machine within the pool of machines available to Condor. Minimal impact on and the security of the remote machine are preserved by Condor through remote system calls. When the job does a system call, for example to do an input or output function, the data is maintained on the machine where the job was submitted. The data is not on the remote machine, where it could be an imposition.

By linking in a set of Condor libraries, system calls are caught and performed by Condor, instead of by the remote machine's operating system. Condor sends the system call from the remote machine to the machine where the jobs was submitted. The system call's function executes, and Condor sends the result back to the remote machine.

This implementation has the added benefit that a user submitting jobs to Condor does not need an account on the remote machine.

1.1.3 Small Businesses Like Condor

Condor starts with the assumption that you have relatively long running tasks that do not require user interaction. While this is not common in small business environments, it does occur. To take examples from businesses that we know are using Condor, tasks involve rendering 3D scenes for a movie, performing a nightly build and regression test on software under development, simulating and analyzing stock market behavior, and simulating the effects of various political decisions. Modern video codecs often take a long time to encode, and any business generating video files could use Condor to manage the process. A small biotechnology company might want to use Condor to manage the long running pattern searches over the human genome. A small engineering company might have similar needs with long running simulations of stress on a building, wind tunnel simulations for cars, or circuit simulations for new electronics devices.

Condor helps those businesses with long running tasks. Such businesses may be using some sort of batch system already, or operate by starting the program each evening, hoping that it finishes before they return in the morning. This is the sort of situation in which Condor excels. Condor also saves time and effort when the time it takes a user to get jobs executing is longer than a few moments, or when a large number of jobs (of any size) must be started.

Condor allows almost any application that can run without user interaction to be managed. This is different from systems like SETI@Home and ProteinFolding@Home. These programs are custom written. Most small companies will not have the resources to custom build an opportunistic batch processing system. Fortunately, Condor provides a general solution.

Condor can be useful on a range of network sizes, from small to large. On a single machine, Condor can act as a monitoring tool that pauses the job when the user uses the machine for other purposes, and it restarts the job if the machine reboots. On a small dedicated cluster, Condor functions well as a cluster submission tool. If you have long running jobs but can not afford to purchase dedicated machines to run the jobs, you can use Condor's opportunistic behavior to scavenge cycles from desktop machines when their users are not using the machines (for example, in the evening or during lunch).

In a typical business these desktop machines are unused for twelve or more hours per day. This processing time is available at no extra cost under Condor. A long running job expected to require the exclusive use of a workstation for two days may be able to produce results overnight.

Condor's functionality called DAGMan, manages the submission of a large number of jobs with simple or complex dependencies on each other. A simple example is that job A and B must complete before job C can start. A rendering example of this would be that job A renders a 3D special effect, job B renders the background, and job C superimposes the special effect onto the background. Condor DAGMan can also be used to run a series of jobs (linearly).

If the small business is using Globus grid resources to gain access to more computing power than it has available in house, Condor-G provides reliability and job management to their jobs. Or, with Condor glidein, remote Globus grid resources can transparently become part of a virtual Condor cluster.

1.1.4 Everyone Benefits

As more machines join a Condor pools, the quantity of computational resources available to the users grows. While Condor can efficiently manage the queuing of jobs where the pool consists of a single machine, Condor works extremely well when the pool contains hundreds of machines.

A contributor to Condor's success is its ClassAd mechanism. Jobs want to find machines upon which they can execute. A job will require a specific platform on which to execute. Machines have specific resources available, such as the platform and the amount of available memory. A separate ClassAd is produced for each job and machine, listing all attributes. Condor acts as a matchmaker between the jobs and the machines by pairing the ClassAd of a job with the ClassAd of a machine.

This mechanism is much more flexible than the simple example of matching the platforms of jobs with those of machines. A job may also prefer to execute on a machine with better floating point facilities, or it may prefer to execute on a specific set of machines. These preferences are also expressed in the ClassAd. Further, a machine owner has great control over which jobs are executed under what circumstances on the machine. The owner writes a configuration file that specifies both requirements and preferences for the jobs. The owner may allow jobs to execute when the machine is idle (identified by low load and no keyboard activity), or allow jobs only on Tuesday evenings. There may be a requirement that only jobs from a specific group of users may execute. Alternatively, any of these may be expressed as a preference, for example where the machine prefers the jobs of a select group, but will accept the jobs of others if there are no jobs from the select group.

In this way, machine owners have extensive control over their machine. And, with this control, more machine owners are happy to participate by joining a Condor pool.

1.2 Condor Code of Conduct

Condor is a terrific tool for performing parametric studies and other type of jobs that can run simultaneously and independently in a number of machines. Nevertheless, under certain circumstances, if you are not careful you can bring the network to a crawl. To avoid these situations, please stick to this simple code of conduct:

- Submit jobs only from your machine or from a machine whose owner you have contacted and is aware of the extra load that you will put on it. No submission from public machines, sorry! (For each Condor running job, there is a process running in the submitting machine, plus lots of network connections, so the submitting machine pays a big toll, which is not fair to pass it to someone else unawares).
- If you plan to run I/O intensive code (i.e. code that reads or writes to disk very large files, or small ones but very often), get in touch with me first. Depending on how I/O intensive your code is, it might not be worth it to use Condor, or I might be able to offer you counsel on how to best do it. Hopefully your Condor submission will perform faster if we take this into account.
- Test your submission. Don't go nuts and submit a 10000 jobs submission without first making sure the whole thing will work with a smaller subset. Start small, verify that things are going OK, check the logs to see that the jobs can access all the necessary files, etc. and only when you are satisfied that things are working go for the big submission.

Please stick to these basic rules so that we can avoid Condor affecting other users' work.

1.3 Manual del usuario de Condor (Adrián Santos Marrero)

1.3.1 ¿Qué es Condor?

Condor es un proyecto de la Universidad de Wisconsin-Madison (UW-Madison). Está ideado para aprovechar al máximo la capacidad computacional de una red de ordenadores. Normalmente sólo disponemos de la potencia del ordenador que estamos usando para ejecutar nuestros trabajos, y si, por ejemplo, tuviéramos que lanzar 100 veces un mismo programa con distinta entrada, tendríamos que hacerlo secuencialmente con la consecuente pérdida de tiempo. Condor nos permite ejecutar nuestro trabajo en tantas máquinas como haya disponibles, por lo que, en el mejor de los casos, nuestro trabajo finalizará en el tiempo que tarda en ejecutarse el más lento de nuestros procesos.

Condor pone a nuestra disposición toda la capacidad de cálculo desaprovechada en nuestra red, de esta manera, los recursos disponibles se incrementan considerablemente.

Condor nos será útil siempre que necesitemos ejecutar un trabajo intenso, tanto computacionalmente como en el tiempo. Al aprovechar solamente recursos ociosos no influye en el uso cotidiano de los ordenadores (ver sección "¿Cómo funciona?").

Además, nos permite:

- Conocer el estado de nuestros trabajos en cada momento
- Implementar nuestras propias políticas de orden de ejecución
- Mantener un registro de la actividad de nuestros trabajos
- Añadir tolerancia a fallos a nuestros trabajos

1.3.2 ¿Cómo funciona?

Básicamente, enviamos un trabajo a Condor, este lo pone en una cola, lo ejecuta y finalmente nos avisa del resultado.
Vamos a verlo un poco más de cerca para intentar comprender como funciona:

- Normalmente usaremos Condor porque queremos ejecutar repetidas veces un programa (posiblemente con diferente entrada) o porque se requiere mucho tiempo para su finalización y, mientras tanto, necesitamos seguir usando nuestra máquina.
- Inicialmente nuestro trabajo no necesita ninguna modificación para ser enviado a Condor. Sin embargo, tenemos que escribir un archivo de descripción del envío (ver sección **??**).
- Una vez enviado a Condor, podemos seguirle la pista a nuestro trabajo con el comando {condor\q} (ver sección 1.4) o mediante un registro de actividad (fichero {Log}).
- Condor realiza periódicamente búsqueda de trabajos nuevos e intenta casarlos con recursos disponibles. Si no hay disponibles, el trabajo se quedará a la espera del próximo ciclo.
- Una vez Condor ha encontrado una máquina capaz de ejecutar el trabajo pendiente, lo envía y empieza la ejecución. Pueden ocurrir varias cosas mientras se está ejecutando un trabajo:
  - Lo más deseable sería que finalizara con éxito. Si esto ocurriera se enviarían las salidas del trabajo a donde haya especificado el usuario y se mandaría un correo electrónico al mismo con un resumen de lo ocurrido.
  - En el caso de que la máquina deje de estar utilizable (porque ha vuelto el usuario o alguno de los motivos explicados más abajo) el proceso deberá abandonarla. Si se estaba ejecutando en el universo ''standard'', se realizaría una imagen del estado actual del proceso ({*checkpoint*}) *(ver sec. 1.3.4) y se finalizaría su ejecución. En el resto de universos, simplemente se instará al trabajo a que finalize su ejecución (para ello se le envía la señal SIGTERM y si, pasado un cierto tiempo, no muere se le envía SIGKILL).*
  - *Otra posibilidad es que el propietario del trabajo haya decidido borrarlo de Condor (ver sección 1.5) con lo que finalizará su ejecución inmediatamente.*

A la hora de enviar nuestro trabajo hemos de tomar algunas precauciones:

- Tenemos que elegir un "universo" adecuado: en la mayoría de los casos nos bastará con el universo "vanilla" (ver sec. 1.3.4).
- Nuestro trabajo ha de ser capaz de ejecutarse en un sistema de procesamiento por lotes:
  - Ha de ser capaz de ejecutarse en "background". No ha de solicitar información interactivamente.
  - Puede usar STDIN, STDOUT y STDERR, pero estos serán archivos en vez de los periféricos habituales (teclado y pantalla).
  - Ha de organizar sus archivos de datos. Por ejemplo, separados por ejecuciones.

Notar que Condor no influye en el uso cotidiano de nuestros ordenadores, ya que solo utilizará máquinas ociosas, o lo que es lo mismo, las que cumplan los siguientes puntos:

- No se está usando el ratón o teclado
- No se está usando la máquina remotamente
- No se está usando para ejecutar ningún otro trabajo.

1.3.3 ¿Cómo lo uso?

Condor está compuesto de varias aplicaciones que nos permiten:

- Enviar trabajos a Condor: {*condor\submit*}.
- *Controlar el estado de nuestros trabajos:* {condor\q}.
- Borrar un trabajo: {*condor\rm*}.
- *Obtener información del estado de Condor:* {condor\status}.

1.3.3.1 Enviando trabajos. {*condor\submit*} Para realizar el envío tenemos que especificar algunas opciones para que Condor sea capaz de manejar adecuadamente nuestro trabajo. Estas opciones incluyen qué comando se va a ejecutar, cuantas veces y con qué entrada, donde irá la salida de cada comando, requisitos de la máquina donde se ha de ejecutar, etc. Esta información se especifica en un fichero de texto que llamaremos fichero de descripción del envío. La sintaxis a seguir la veremos a continuación.

1.3.3.1.1 Fichero de descripción del envío Este archivo será la entrada al comando {*condor_submit*}. *Un ejemplo de fichero de envío se puede ver en el siguiente ejemplo:*

```
############################
#
# foo.submit
#
# Ejemplo 1: Archivo simple de descripción del envío.
#
############################

Executable   = foo
Universe     = vanilla
input        = test.data
output       = foo.out
error        = foo.err
Log          = foo.log
Queue
```

Una vez guardado este fichero, le indicamos a Condor que lo ejecute de la siguiente manera:

```
[adrians@trevina ~]$ condor_submit foo.submit
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 3.
```

Veamos con más detalle el contenido del archivo:

- ***Executable**: Especificamos la ruta y el nombre del archivo ejecutable. En el ejemplo solo se ha especificado el nombre, por lo que se espera que {*`foo`*} y {*`foo.submit`*} estén en el mismo directorio.*
- `Universe: Elegimos un universo, por defecto se usará el universo ‘‘vanilla’’. Ver sección 1.3.4.}`
- `input: Archivo desde donde se leerá la entrada por defecto (stdin). Si no se especifica, se utilizará el archivo {/dev/null}.`
- `output: Archivo donde se escribirá la salida del comando (stdout). Si no se especifica, se utilizará el archivo {/dev/null}.`
- `error: Archivo donde se escribirá la salida de error del comando (stderr). Si no se especifica, se utilizará el archivo {/dev/null}.`
- `Log: Archivo donde Condor almacenará un histórico de lo que le ha ocurrido a nuestro trabajo e información como su código de salida, errores relacionados con Condor, etc.`
- `Queue: Indica que Condor va a ejecutar una vez este trabajo, podemos especificar un número (por ejemplo {Queue 10} o escribir varias veces {Queue} con lo que se ejecutará tantas veces como hayamos escrito). Podemos especificar opciones para cada ejecución, por ejemplo: podemos tener un fichero de entrada ({input}) para cada ejecución de nuestro trabajo.`

Veamos ahora otro ejemplo, esta vez un poco más complicado:

```
############################
#
# complex.submit
#
# Ejemplo 2: Archivo de descripción del envío usando
# Requirements y Rank.
#
############################

Executable   = complex
Universe     = vanilla
Requirements = Memory >= 64 && OpSys == "Linux" && Arch == "INTEL"
Rank         = Memory
input        = data.$(Process)
output       = out.$(Process)
error        = err.$(Process)
Log          = complex.log
Queue 10
```

En este ejemplo introducimos algunas opciones nuevas:

Requirements: *Especificamos los requisitos que se han de cumplir para que nuestro trabajo se ejecute. En el ejemplo obligamos a que la máquina candidata tenga un procesador INTEL o compatible, esté ejecutando Linux y, además, no permitimos que tenga menos de 64MB de memoria RAM. En el caso de que no especifiquemos explícitamente los requisitos sobre arquitectura y sistema operativo, Condor los creará automáticamente para que nuestros trabajos se ejecuten en máquinas con la misma arquitectura y el mismo sistema operativo que la máquina desde donde se envió el trabajo. Para ver los requisitos finales de nuestro trabajo (una vez enviado a Condor) podemos ejecutar* `condor_q -l`*, este comando nos mostrará información detallada de cada trabajo enviado.*

Rank: *Define un valor numérico que expresa preferencia, es decir, dadas todas las máquinas candidatas a ejecutar nuestro trabajo, Condor evalúa esta expresión en cada una de ellas y elegirá aquellas donde su valor sea mayor. En el ejemplo, preferimos aquellas máquinas que tengan mayor cantidad de memoria RAM.*

*Para obtener más información acerca del uso de {*Requirements} y {Rank} vea la {sección 2.5.2}{http://goya/inves/SINFIN/Condor/v6.6/2_5Submitting_Job.html\#SECTION00352000000000000000} del manual de Condor.

En el ejemplo 2 también hemos usado unos nombres de archivo un tanto especiales en las opciones {input}, {output} y {error}. El uso de la cadena ‘‘$\backslash(Process)''implicaquealldondeaparezcasersustituidoporelnmerodeltrabajoqueseva$

1.3.4 Universos

Para Condor, un ‘‘universo’’ define un conjunto de características que marcarán el entorno de ejecución de nuestro trabajo. Por ejemplo, para trabajos en Java existe un universo ‘‘Java’’. Este, por ejemplo, permitirá capturar las excepciones de la máquina virtual de Java o solo ejecutará los trabajos en máquinas con la máquina virtual disponible.

Básicamente podemos elegir entre tres universos:

vanilla Es el universo por defecto y, además, es el menos restrictivo con los trabajos que se pueden enviar (acepta cualquier programa escrito en cualquier lenguaje). La parte negativa es que no permite ‘‘checkpointing o llamadas al sistema remotas (ver universo ‘‘standard’’ a continuación).

standard Este universo soporta ‘‘checkpointing’’ y llamadas al sistema remotas. Hacer ‘‘checkpointing’’ de un programa significa guardar en disco su estado actual de ejecución antes de parar el proceso. Gracias al ‘‘checkpointing’’, un programa se puede parar (guardándose su estado en un fichero), y más adelante se puede volver a ejecutar desde el punto exacto en que se abortó. Para que un programa pueda ser enviado a este universo ha de estar enlazado con las librerías de Condor (compilado usando *condor_compile*). Presenta algunas restricciones en los trabajos que se pueden enviar.

java Este universo está destinado para el envío de trabajos escritos en Java.

Para información más detallada acerca de cada universo puede visitar la sección 2.4.1 del manualhttp://goya/inve

1.3.5 Sobre el acceso a los ficheros

El único universo que dispone de un sistema de llamadas al sistema remotas es el ‘‘standard’’, debido a esto, cuando use otro universo (por ejemplo el ‘‘vanilla’’) asegúrese de que los archivos de entrada y salida de sus trabajos se escriban o lean en directorios compartidos por NFS (es decir, visibles desde todas las máquinas). Un buen ejemplo es su directorio home ({*home/{*user}...}) ya que desde cualquier máquina se puede acceder a él. Un ejemplo de un directorio no compartido sería {*tmp*}, si crea un directorio {*tmp/my*$\backslash_{condor}\backslash_{job}$}, este solo será visible desde su máquina, por lo que cuando su trabajo se empiece a ejecutar en otra máquina y vaya a abrir un archivo en ese directorio se encontrará que no existe y no podrá continuar (estos errores aparecerán en el {Log} del trabajo).

1.4 {

Estado de los trabajos enviados. {*condor*$\backslash_q$*}}*

*Podemos obtener información acerca de nuestros trabajos con el comando {*condor$\backslash_q$}:

```
[adrians@trevina ~]$ condor_submit myjob.submit
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 1.

[adrians@trevina ~]$ condor_q


-- Submitter: trevina.iac.es : <161.72.81.178:1085> : trevina.iac.es
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
   1.0   adrians         7/13 12:37   0+00:00:00 I  0   0.0  myprog Example.1.0

1 jobs; 1 idle, 0 running, 0 held
```

Por defecto, este comando nos mostrará información de los trabajos que hemos enviado desde la máquina donde se ejecuta, en el ejemplo sería ‘‘trevina’’. La información que aparece en la salida sería:

+ El ID es el identificador del trabajo y está formado por dos números:+ El número antes del punto representa el ‘‘cluster’’. Un ‘‘cluster’’ es el conjunto de trabajos creado en un envío. Podemos ver un ejemplo en la salida del comando ‘‘condor_submit’’. + El número después del punto representa el trabajo dentro del cluster, como en el ejemplo solo creamos uno, será el trabajo 0. Trabajos sucesivos en el mismo cluster se nombrarían como 1.1, 1.2, ...

- El usuario que envío los trabajos.
- La fecha del envío.
- El tiempo de ejecución, en el formato: Días+HH:MM:SS.
- El estado actual del trabajo. Algunos valores posibles son:

  I: No se está ejecutando porque aun no se le ha asignado ninguna máquina (IDLE).

  R: Ejecutándose actualmente (RUNNING).

  H: El trabajo no se está ejecutando por deseo del propietario

  (HOLD). Ver el comando condor_hold {http://goya/inves/SINFIN/Condor/v6.6/condor_hold.html}, condor_relea {http://goya/inves/SINFIN/Condor/v6.6/condor_release.html} o la sección 2.6.3http://goya/inves/SINFIN/Co del manual de Condor.

- La prioridad del trabajo que ha especificado el usuario. Ver comando

condor_priohttp://goya/inves/SINFIN/Condor/v6.6/condor_prio.html o la sección 2.6.4http://goya/inves/SINFIN/Co del manual de Condor.

- El tamaño de la imagen del trabajo en megabytes.
- Y por último, el nombre del ejecutable.

En el caso de que un trabajo no se esté ejecutando, este comando también nos permite conocer el motivo gracias a la opción {-analyze}. Por ejemplo: %Este comando también nos permite conocer las causas de que un trabajo no se esté %ejecutando, para esto usaremos la opción {-analyze}. Por ejemplo:

```
[adrians@trevina ~]$ condor_submit myjob.submit
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 1.

[adrians@trevina ~]$ condor_q -analyze


-- Submitter: trevina.iac.es : <161.72.81.178:39869> : trevina.iac.es
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
---
001.000:  Run analysis summary.  Of 187 machines,
    187 are rejected by your job's requirements
      0 reject your job because of their own requirements
      0 match, but are serving users with a better priority in the pool
      0 match, but prefer another specific job despite its worse user-priority
      0 match, but will not currently preempt their existing job
      0 are available to run your job
        No successful match recorded.
        Last failed match: Thu Sep 16 12:38:09 2004
        Reason for last match failure: no match found

WARNING:  Be advised:
   No resources matched request's constraints
   Check the Requirements expression below:

Requirements = ((Memory > 2147483647)) && (Arch == "INTEL") &&
(OpSys == "LINUX") && (Disk >= DiskUsage) &&
(TARGET.FileSystemDomain == MY.FileSystemDomain)
```

En el ejemplo podemos ver como el trabajo 1.0 tiene problemas para ejecutarse: nuestros requisitos ({Requiremen han desechado 187 máquinas de las 187 candidatas. Además, {*condor_q*} *nos sugiere que revisemos dicha expresión y nos la muestra en su salida (en el ejemplo vemos como el límite mínimo de memoria RAM es excesivo).*

*Para más información puedes visitar la {página del manual de {*condor_q}} {http://goya/inves/SINFIN/Condor/v6.6/condor_q.html}, la sección 2.6.1http://goya/inves/SINFIN/Condor/v6.6/2_6Managing_Job.html#SECTION00361000000 o la sección 2.6.5http://goya/inves/SINFIN/Condor/v6.6/2_6Managing_Job.html#SECTION00365000000000000000 del manual de Condor.

1.5 {

Borrando trabajos. {*condor\rm*}}

Usaremos {condor\rm} para borrar un trabajo de la cola de Condor:

```
[adrians@trevina ~]$ condor_q


-- Submitter: trevina.iac.es : <161.72.81.178:1085> : trevina.iac.es
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
   2.0   adrians         7/13 12:46   0+00:00:01 R  0   0.0  myprog Example.2.0

1 jobs; 0 idle, 1 running, 0 held

[adrians@trevina ~]$ condor_rm 2.0
Job 2.0 marked for removal
```

Podemos especificar tanto un trabajo como un cluster, en el ejemplo anterior, si hubiésemos ejecutado {condor\rm 2} habríamos borrado todos los trabajos del cluster 2.

Notar que no podemos borrar trabajos que no nos pertenezcan.

Para más información puede visitar la {página del manual}{http://goya/inves/SINFIN/Condor/v6.6/condor_rm.html} o la {sección 2.6.2 del manual de Condor}{http://goya/inves/SINFIN/Condor/v6.6/2_6Managing_Job.html\#SECTION00362000000000000000}.

1.6 {

Estado de Condor. {*condor\status*}}

Este comando nos permitirá conocer el estado de Condor:

```
[adrians@trevina ~]$ condor_status

Name           OpSys       Arch   State      Activity   LoadAv Mem   ActvtyTime

vm1@aciano.ia LINUX       INTEL  Claimed    Busy       1.030   501  0+16:56:02
vm2@aciano.ia LINUX       INTEL  Claimed    Busy       0.990   501  0+00:59:48
agracejo.iac. LINUX       INTEL  Claimed    Busy       1.030   500  0+21:00:39
vm1@agrimonia LINUX       INTEL  Claimed    Busy       1.010  1826  0+00:09:36
vm2@agrimonia LINUX       INTEL  Claimed    Busy       1.000  1826  0+00:09:32
alfalfa.iac.e LINUX       INTEL  Owner      Idle       0.000   248  0+00:32:55
...
tonina.iac.es SOLARIS29   SUN4u  Claimed    Busy       1.000   128  0+21:56:24
toro.iac.es   SOLARIS29   SUN4u  Unclaimed  Idle       0.000   128  0+00:00:04
tuno.iac.es   SOLARIS29   SUN4u  Owner      Idle       0.040   640  0+01:33:15
vibora.iac.es SOLARIS29   SUN4u  Claimed    Busy       1.010   576  3+02:59:06
viola.iac.es  SOLARIS29   SUN4u  Claimed    Busy       1.010   256  0+01:40:35
zarza.iac.es  SOLARIS29   SUN4u  Claimed    Busy       0.660   256  0+00:01:06
zorro.ll.iac. SOLARIS29   SUN4u  Claimed    Busy       1.040   384  1+03:38:25

                     Machines Owner Claimed Unclaimed Matched Preempting

          INTEL/LINUX        75    33      41         1       0          0
    SUN4u/SOLARIS29          87    21      64         2       0          0

                 Total      162    54     105         3       0          0
```

Este comando muestra información sobre cada una de las máquinas que forman el "pool" de Condor y un resumen del estado actual. En este resumen podemos comprobar, en cifras, el uso que se le están dando a las máquinas. Así, por ejemplo, podremos comprobar cuantas máquinas quedan disponibles para ejecutar trabajos mirando la columna "Unclaimed".

Para más información puedes visitar la página del manual {http://goya/inves/SINFIN/Condor/v6.6/condor_status.html} de este comando.

%%%%%%%%%%%%%%%%%%%%%%%% Section.

2 Limitaciones

+ Si su trabajo realiza muchas operaciones de entrada/salida (E/S) tenga en cuenta la sobrecarga que esto conlleva, probablemente no obtenga una mejora muy grande enviándolo a Condor. Note que todas

las operaciones de lectura/escritura sobre archivos se realizan sobre la red[1] *por lo que sus trabajos se verán ralentizados. + Si envía un trabajo al universo "vanilla" contemple que cuando sea expulsado de una máquina perderá todo lo procesado hasta ese momento (a no ser que haya tomado precauciones por su cuenta). Si envía un trabajo que planea que vaya a tardar varios días en finalizar su ejecución probablemente no obtenga mejoría usando Condor, en estos casos plantéese el uso del universo "standard". + Tenga en cuenta que cada trabajo que envíe generará un proceso "shadow" en la máquina desde donde se hace el envío. Este proceso se encarga de resolver algunas cuestiones relacionadas con su trabajo, realmente no consume demasiada CPU pero si realiza muchos envíos puede llegar a ralentizar su máquina.*

%%%%%%%%%%%%%%%%%%%%%%%% Section.

3 ¿Y si tengo problemas?

Existe a su disposición un portal dedicado a Condor en el I.A.C., la dirección es: http://goya/inves/SINFIN/Condor/. También puede ponerse en contacto con los administradores de Condor en la dirección de correo condor@iac.es.

3.1 Python bindings

[1] Los archivos residen físicamente en el home de un usuario especial por lo que todas las peticiones se realizarán sobre NFS.