ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

"Estudio, implementación y evaluación de entornos de computación de alto rendimiento HTC"

AUTOR: María Rosa Liarte López
DIRECTOR: Mª Victoria Bueno Delgado

Diciembre / 2007

Autor	María Rosa Liarte López
E-mail del Autor	rosarosike@hotmail.com
Director(es)	Mª Victoria Bueno Delgado
E-mail del Director	Mvictoria.bueno@upct.es
Codirector(es)	
Título del PFC	"Estudio, implementación y evaluación de entornos de computación de alto rendimiento HTC"
Descriptores	

Resumen

Para llevar a cabo el procesamiento de tareas con un cómputo muy grande se suele recurrir a lo que hoy día llamamos "supercomputadoras" que, con una capacidad de procesamiento muy superior a la de los ordenadores de sobremesa, son capaces de ejecutar cientos de tareas en un tiempo relativamente corto. El principal problema de estos "superordenadores" es el coste económico, ya que es este hardware presenta un precio muy elevado que no todas las empresas pueden abordar. Para solventar el obstáculo económico, en los últimos años han comenzado a utilizarse soluciones más rentables: los llamados entornos de alto rendimiento computacional HTC (High-Throughput Computing).

Tras comprobar como un entorno HTC permite aprovechar los recursos de los que dispone una red cuando se detecta inactividad en ésta, se describe en este proyecto la implementación de un entorno HTC prototipo para una futura implementación en los laboratorios de Área de Ingeniería Telemática. Este entorno permitirá al personal docente e investigador utilizar los recursos de los que dispone el área para poder realizar sus tareas de investigación de una forma sencilla y cómoda a través de una interfaz web realizada en un proyecto paralelo a este.

Titulación	Ingeniería Técnica de Telecomunicaciones, especialidad Telemática
Intensificación	
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Diciembre 2007

Índice

Índice figuras y tablas

Capítulo 1
Introducción

1.1 Antecedentes

Hoy en día, la mayoría de las empresas dedicadas a I+D+I así como las universidades necesitan simular y ejecutar numerosas tareas con el fin de obtener ciertos resultados. Algunas de estas tareas pueden consumir millones de ciclos de computación, lo que implica tiempos muy elevados de ejecución. Para solventar este problema se recurre desde hace años a las llamadas "supercomputadoras" que, con una capacidad de procesamiento muy superior a la de los ordenadores de sobremesa, son capaces de ejecutar cientos de tareas en un tiempo relativamente corto. El principal problema de estos "superordenadores" es el coste económico, ya que es este hardware presenta un precio muy elevado que no todas las empresas pueden abordar.

Para solventar el obstáculo económico, en los últimos años han comenzado a utilizarse soluciones más rentables: los llamados entornos de alto rendimiento computacional HTC (*High-Throughput Computing*). Estos entornos están formados principalmente por una gran cantidad de ordenadores de sobremesa puestos en red, con el mismo o distinto sistema operativo. El objetivo de estos entornos es aprovechar los ciclos de procesamiento de los ordenadores cuando no se detecta actividad en ellos para poder ejecutar tareas previamente lanzadas por un usuario a un ordenador central de ese entorno. Este ordenador central se encarga de monitorizar la actividad del resto de ordenadores repartiendo las tareas que le llegan entre los ordenadores que no presentan actividad durante un periodo de tiempo. De esta forma se aprovechan los ciclos de computación de aquellas redes que, con los ordenadores encendidos, no presentan ninguna actividad, para poder utilizarlos emulando una "supercomputadora". Se deduce pues que la solución de un entorno distribuido HTC no es una tarea fácil de implementar debido a su complejidad a nivel software pero conlleva un gran ahorro económico.

La Universidad Politécnica de Cartagena, y más concretamente en el Área de Ingeniería Telemática dispone hoy en día de varias "supercomputadoras" o centros de procesamiento para poder lanzar las tareas que realiza el personal docente e investigador de una forma sencilla contribuyendo, por un lado, a la rápida respuesta de estas máquinas en tiempo de procesamiento y evitando así que el investigador gaste los recursos de su ordenador personal pudiendo, de esa forma, utilizarlos para otros fines. Algunos aspectos técnicos de estas supercomputadoras se detallan a continuación:

- "Sísifo", (sísifo.upct.es) máquina del Área de Ingeniería Telemática, está compuesta por una máquina Sun, modelo v40z con 4 procesadores de 2 núcleos AMD, modelo 880-885, a 2,6 GHz. El rendimiento teórico de esta máquina es de 38,4 Gflops.
- "Prometeo", (prometeo.sait.upct.es) máquina del Servicio de Apoyo a la Investigación Tecnológica (SAIT), es un sistema paralelo de memoria distribuida con un total de 16 procesadores a 1GHz, con 8 MB de caché

de nivel 2, 16Gbytes de memoria y unos 300Gbytes de almacenamiento en disco. Su rendimiento teórico es de 32Gflops.

Hay que destacar que la máquina de Prometeo no es de uso exclusivo del área, sino que la utilizan todos los investigadores de la Universidad Politécnica de Cartagena, lo que implica compartir los recursos de dicha máquina, algo que, en ocasiones, puede dar como resultado una larga espera a la hora de obtener resultados de las tareas lanzadas en esta máquina.

Por otro lado, el Área de Ingeniería Telemática dispone de seis laboratorios docentes y tres laboratorios de investigación (I+D) donde se contabilizan un total de 101 máquinas entre servidores, routers y PCs. En la siguiente tabla se muestra las características técnicas de estas máquinas en términos de procesamiento. Además se detalla un estudio realizado sobre el número medio de horas a la semana que se utilizan estos laboratorios.

Laboratorio	Nº Servidores / Router	Procesador Servidor / Router	Nº PCs	Procesador PCs	Nº horas uso a la semana	% uso a la semana	Nº horas libre a la semana	% tiempo libre a la semana
IT-1	1	450 MHZ	15	Core 2 duo 2,13 GHZ	10	5.95%	158	94.04%
IT-2	1	Core 2 duo 2,13 GHZ	15	Core 2 duo 2,13 GHZ	14	8.33%	154	91.66%
IT-3	1	2,4 GHZ	10	2,4 GHZ	21	12.5%	147	87.5%
IT-4	1	1,5 GHZ	7 2	1 GHZ 2,4 GHZ	8	4.76%	160	95.23%
IT-5	1	Core 2 duo 2,13 GHZ	10	Core 2 duo 2,13 GHZ	14	8.33%	154	91.66%
IT-6	1 1	2 Xeon 3,8 GHZ 2 UltraSPARC 2 400 MHZ	10	530 MHZ	18	10.71%	150	89.28%
I+d+I-1	X		1 5	Core 2 due 2,13 GHZ Core 2 Duo 500 MHZ	12	7.14%	156	92.85%
I+d+I-2	1	Dual Core 2,8 GHZ	7 1	3 GHZ 4,3 GHZ	12	7.14%	156	92.85%
I+d+I-3	X		2 4 1 1	Core 2 duo 2,13 GHZ Core 2 duo 1,86 GHZ 2,8 GHZ 2,4 GHZ	12	7.14%	156	92.85%

Tabla 1. 1 Estudio rendimiento computacional del AIT (I)

Realizando unos sencillos cálculos se puede obtener el rendimiento teórico en FLOPS (*Floating Operation Point per Second*) que ofrecen las 101 máquinas. Para ello hay que tener en cuenta que todos los procesadores de los laboratorios son de arquitectura Intel. Según la página oficial de Intel el rendimiento teórico de sus procesadores de un único núcleo (*Single Core*) es de 8 flops por hercio, mientras que los procesadores de doble núcleo (*Dual Core*) es de 10 flops por hercio. Según estas especificaciones se obtiene el siguiente rendimiento:

Procesador GHZ	Gflops Teóricos	PCs Single Core	PCs Dual Core	Gflops Totales
0,45	3,6	1	-	3,6
0,5	5	-	5	25
0,53	4,24	10	-	42,4
1	8	7	-	56
1,5	12	1	-	12
1,86	18,6	-	4	74,4
2,13	21,3	-	44	937,2
2,4	19,2	14	-	268,8
2,8	22,4/28	1	1	50,4
3	24	7	-	168
4,3	34,4	1	-	34,4
	Gflops Totales = 1672,2			

Tabla 1. 2 Estudio del rendimiento computacional del AIT (II)

Los cálculos que se han realizado no se han tenido en cuenta los servidores del laboratorio IT-6 ya que estos se utilizan para otros fines. Por tanto, se tienen un total de 99 máquinas que ofrecen un rendimiento computacional de 1672,2 Gflops.

Hay que tener en cuenta que estos PCs no están siempre disponibles, asi pues, a continuación se va a realizar una comparativa del rendimiento de los mismos frente a las dos "supercomputadoras" de las que dispone el área para conocer de una forma más exacta los ciclos de computación que el área está desaprovechando. Para ello, se asume un total de 12 horas diarias (de 9:00 p.m. a 9:00 a.m.) en las que los 99 PCs no están siendo utilizados. El número total de Operaciones de punto flotante que se podrían realizar durante únicamente una semana son:

Total FLOPS = $1672,2*10^9$

Total segundos en una semana (solo 12 horas al día) = 60*60*12*7 = 302400 s.

Nº operaciones totales a la semana (FLOP) = $\frac{1672,2*10^9}{302400} = 5,52*10^6$

Para el caso de prometeo, y suponiendo uso exclusivo solo para el Área de Ingeniería Telemática

Total FLOPS = $32*10^9$

Total segundos en una semana (solo 24 horas al día) = 60*60*24*7 = 604800 s.

Nº operaciones totales a la semana (FLOP) = $\frac{32*10^9}{640800} = 52,91*10^3$

Para el caso de Sísifo, el número de operaciones totales a la semana, obtenido siguiendo las mismos criterios que para prometeo son:

Nº operaciones totales a la semana (FLOP) = $\frac{38,4 * 10^9}{640800} = 59,92 * 10^3$

Tras la comparativa, se demuestra como la suma de los ciclos de computación de las máquinas que forman parte del Área de Ingeniería Telemática es muy superior al de las supercomputadoras que hoy por hoy pueden acceder los investigadores del área para lanzar sus tareas. Por tanto existe un claro desaprovechamiento de ciclos de computación de las máquinas de los laboratorios del área.

1.2 Objetivos

Tras comprobar como un entorno HTC permite aprovechar los recursos de los que dispone una red cuando se detecta inactividad en ésta, se plantea como objetivo realizar un entorno HTC prototipo para una futura implementación en los laboratorios de Área de Ingeniería Telemática. Este entorno permitirá al personal docente e investigador utilizar los recursos de los que dispone el área para poder realizar sus tareas de investigación de una forma sencilla y cómoda a través de una interfaz web realizada en un proyecto paralelo a este.

Para llevar a cabo este proyecto será necesario, primero, dar respuesta diversas cuestiones respecto a las necesidades del usuario final y solventarlas teniendo en cuenta las restricciones impuestas en cuanto al hardware y el software disponible en dichos laboratorios. A nivel instalación, se resolverán, entre otras, las siguientes cuestiones:

- ¿Qué tipo de entorno HTC se adapta más a las necesidades del prototipo “usuario final” y al hardware y software del que se dispone?
- ¿El sistema operativo que actualmente reside en los ordenadores de los laboratorios influye en dicho entorno?
- ¿Será necesario tener algún software adicional instalado en los ordenadores de los laboratorios para llevar a cabo la instalación del entorno?
- ¿La existencia de este entorno HTC en los laboratorios implica una reducción del rendimiento de los ordenadores cuando estos los utilicen los alumnos en su formación?
- ¿Se podría ampliar la instalación de este entorno al resto de laboratorios de la Escuela de Telecomunicaciones?

A nivel usuario, deberán responderse, entre otras, las siguientes preguntas:

- ¿El usuario necesitará realizar alguna instalación de software adicional en su ordenador personal para poder trabajar con el entorno?
- ¿Un usuario podrá acceder de forma simple y sencilla al entorno HTC?
- ¿Se podrán lanzar todo tipo de tareas, independientemente del lenguaje en el que hayan sido escritas?

Una vez solventadas esas cuestiones, el objetivo será la instalación, configuración y puesta en marcha del entorno adecuado realizando diversas pruebas para comprobar el perfecto funcionamiento del mismo y añadiendo funciones adicionales que se consideren útiles para un futuro próximo.

1.3 Estructura del contenido

El contenido del proyecto se define en los siguientes capítulos:

En el capítulo 2 se presentan los distintos entornos computacionales que se utilizan hoy en día y se justifica la elección del entorno computacional HTC Condor. Se describe el entorno, características principales, arquitectura y modos de funcionamiento, etc..

En el capítulo 3 se explican los pasos necesarios para la instalación de dicho entorno bajo los sistemas operativos Windows XP y Unix con la distribución RedHat. Se describe el escenario en el que se realiza la instalación así como los requisitos hardware y software. Por último se detalla la configuración necesaria para poner en marcha el entorno.

En el capítulo 4 se describen los procesos y tareas que se pueden lanzar en el entorno de computación instalado. También se describen los requisitos a cumplir y pasos a seguir para poder lanzar las tareas, para la ejecución y por último, la obtención de resultados.

En el capítulo 5 se detallan las conclusiones del trabajo realizado y las posibles líneas futuras.

Por último se añade un apéndice para explicar diversas configuraciones adicionales que se pueden implementar en el entorno para dar una mayor funcionalidad al mismo.

Capítulo 2
Entornos de Computación. La elección de Condor

2.1 Introducción a los entornos de computación.

Hace años, los científicos y los ingenieros dependían del uso de una máquina central, una supercomputadora, para que realizara el trabajo computacional. Un gran número de individuos y grupos necesitaban unir sus recursos financieros, para poder permitirse una de estas máquinas. Los usuarios debían de esperar su turno, y además tenían un espacio limitado de tiempo para llevar a cabo su tarea.

En el momento que los ordenadores se volvieron más pequeños, mas rápidos y baratos, los usuarios se distanciaron del uso de las supercomputadoras, y compraron sus propias estaciones de trabajo y PC´s. Un individuo o grupo pequeño podrían permitirse recursos computacionales suficientes para lo que se necesitase. “El PC es más pequeño que las supercomputadoras, pero posee acceso exclusivo”.

Para muchas investigaciones y proyectos de ingeniería, la calidad de las investigaciones o del producto es fuertemente dependiente con la cantidad de ciclos de ordenador que se permitan. Científicos e ingenieros que trabajan en esta clase de proyectos necesitan un entorno de ejecución que reparta gran capacidad de poder computacional a lo largo de un gran período de tiempo. Estos entornos de ejecución es lo que hoy en día se denominan “Entornos de computación”. Dentro de los entornos de computación hay que distinguir entre distintos tipos, los cuales se detallan en las siguientes secciones.

2.1.1 Entornos HPC

El término HPC (*High Performance Computing*) se refiere al uso de supercomputadoras (paralelas) y clústers de máquinas, esto es, sistemas computacionales formada por múltiples procesadores (normalmente de gran potencia), unidos en un sistema único, con conexiones comerciales disponibles. Esta arquitectura entra en contraste con las máquinas del tipo *Mainframe*, que son generalmente únicas por naturaleza. Mientras que tareas de cierto nivel necesitan usar este tipo de sistemas, pueden ser creados a partir de componentes separados. Debido a su flexibilidad, poder, y relativo bajo coste, los sistemas HPC han dominado de manera incremental el mundo de las supercomputadoras. Normalmente, los sistemas de ordenadores cercanos a la región de los tera-flops, son considerados ordenadores HPC.

Los sistemas HPC son mayoritariamente usados para investigaciones científicas. Un término relacionado HPTC (*High-Performance Technical Computing*), generalmente se refiere a aplicaciones del mundo de la ingeniería basadas en clúster (como los cálculos fluidomecánicos y el estudio virtual de los prototipos).

De manera reciente, HPC ha empezado a usarse para uso comercial, basado en clústers de supercomputadoras, como por ejemplo las data warehouse, aplicaciones line-on-business (LOB), y procesamiento de transacciones.

2.1.2 Entornos GRID

Desde que los ordenadores fueron conectados en red, la idea del Grid Computing ha estado latente, y no había progresado debido principalmente a la gran variedad técnica de la industria informática: múltiples sistemas operativos, arquitecturas de procesadores, lenguajes de programación, protocolos de red, etc. Sin embargo, debido a la perseverancia de sus seguidores, la omnipresencia de Internet y la casi ubicuidad de Windows, esta tecnología está haciéndose realidad.

El Grid Computing, es la tecnología que consta de una infraestructura que permite el acceso y procesamiento concurrente de un programa, entre varias entidades computacionales independientes, que actúan como un único gran sistema. Se usa normalmente para programas que requieren procesos de gran escala y/o acceso a mucha cantidad de datos.

Entre las características principales que distinguen al Grid Computing podemos citar las siguientes:

- Permite integrar sistemas y dispositivos heterogéneos, pues permiten que recursos diferentes puedan interactuar entre sí.
- Mejora del coste efectivo de los entornos operativos, pues permite aprovechar al máximo los recursos disponibles en una red, y de esta manera a su vez mejora la capacidad de los recursos para responder a las fluctuaciones de la demanda.
- Las tecnologías grid son flexibles, pues son capaces de ajustarse dinámicamente a los entornos cambiantes y fluctuantes de las tecnologías de la información.
- Aumenta la fiabilidad de la infraestructura, sacando ventaja de los recursos del grid como una alternativa ante la recuperación de los desastres tradicionales.

Los objetivos que persigue el Grid Computing para una empresa u organización se citan a continuación:

- Mejorar los tiempos para la producción: Pues permite incrementar la productividad y colaboración; y de esta manera las organizaciones mejoran sus tiempos de resultados y por lo tanto rapidez en el tiempo de lanzamiento al mercado, que en última instancia constituye una ventaja competitiva.
- Permitir la colaboración y promover flexibilidad operacional: Pues no solo unirá recursos tecnológicos dispares, sino también gente y aptitudes; permitiendo de esta manera la posibilidad de compartir, acceder y gestionar información, mejorando la colaboración entre unidades empresariales.
- Escalar para satisfacer demandas variables del negocio: Permite crear infraestructuras operativas flexibles y resistentes, que faciliten abordar rápidas fluctuaciones en la demanda, accediendo instantáneamente a recursos de computación y datos para "sentir y responder" a las necesidades de negocio.
- Incremento de la productividad: Dando a los usuarios finales acceso a los recursos de computación, datos y almacenamiento que necesiten y cuando

los necesiten, ayudando a las empresas a equipar mejor a sus empleados para efectuar sus tareas, resolver problemas comerciales complejos con facilidad y moverse entre etapas del diseño de productos, proyectos de investigación y más, todo más rápidamente.

- Aprovechar inversiones de capital existentes: Maximizar la utilización eficiente y productiva de los recursos existentes es una de las claves para reducir costes operativos. Además, las empresas pueden aprovechar los recursos grid para entregar escenarios de back up y recuperación efectivos y de bajo coste, sin necesidad de invertir para duplicar sistemas.

Sin embargo, y a pesar de las ventajas, el objetivo del proyecto es el uso optimizado de los recursos de los laboratorios de la facultad de telecomunicaciones, por lo que la principal característica de los entornos Grid, que es la capacidad de compartir recursos de diferentes zonas, se desaprovecha.

Además, y como se verá en las siguientes secciones, el entorno de computación elegido (HTC con la herramienta Condor) permite en su configuración el Universo Grid a partir del entorno HTC de Condor. Todo ello se realiza mediante el mecanismo de Flocking, cuyo funcionamiento se explica de forma más ampliada en el apéndice de este proyecto.

2.1.3 Entornos HTC

La clave de los entornos HTC (como Condor) es que aprovechan el uso de todos los recursos disponibles. Existen muchas ventajas para que se haya decantado por un entrono HTC, alguna de las cuales se citan a continuación:

- HPC puede ser usado para soportar decisiones o aplicaciones de tiempo limitado. Sin embargo para hacer análisis altamente sensibles, estudios o simulaciones que necesitan para su ejecución largos períodos de tiempo, se necesita un mecanismo como HTC.
- El software que opera con un mecanismo de alto rendimiento computacional HTC, en lugar de HPC, es capaz de organizar las máquinas en clústers o colecciones de clústers.
- Los entornos de computación HPC en ocasiones con descritos en términos de operaciones en coma flotante por segundo (FLOPS). Muchos científicos no tratan hoy en día las tareas FLOPS, ya que los problemas que esto trae consigo son de gran importancia. Las tareas en las que se centran los investigadores con aquellas que ocupan un largo período de tiempo de procesamiento para su resolución y un alto rendimiento de procesamiento, es decir, las tareas FLOPY (operaciones en coma flotante por año) bajo el entorno HTC.

2.2 Condor: entorno de computación masiva.

2.2.1 Introducción

Condor es un entorno de computación HTC, un “sistema conjunto” de máquinas que dirige tareas de gran cómputo. Como muchos sistemas conjuntos, Condor posee un mecanismo de colas, políticas de preferencia, esquemas de prioridad, y clasificaciones de los recursos. Los usuarios lanzan sus tareas en Condor, Condor pone la tarea en la cola, las ejecuta, e informa a los usuarios del resultado. Estos sistemas, normalmente operan sólo en máquinas dedicadas. Con frecuencia, estas

máquinas pertenecen a una organización, y sólo se dedican a la ejecución de tareas para dicha organización. Condor puede fijar tareas en máquinas dedicadas.

A pesar de los sistemas conjuntos normales, Condor también está diseñado para usar máquinas no dedicadas para ejecutar tareas. Indicando que sólo ejecuten tareas en máquinas que no están siendo usadas, Condor puede de manera efectiva usar máquinas que estén en espera en un conjunto (a partir de este momento se denominará Pool) de máquinas. Esto es importante, porque a menudo la capacidad de cómputo representado por la adicción total de todas las estaciones de trabajo no dedicadas a través de un Pool, es mucho mayor, que el poder de cómputo de un recurso central dedicado, como se pudo comprobar en la introducción de este proyecto.

2.2.2 Características principales

La funcionalidad básica de Condor es la siguiente: un usuario lanza una tarea sobre Condor. Condor busca una máquina adecuada en su red, y comienza a ejecutar la tarea en dicha máquina. ¿Y que pasa si por ejemplo, el dueño de la máquina donde se está ejecutando una tarea vuelve a su puesto de trabajo y comienza a utilizar el teclado?.Condor tiene la capacidad de detectar si una máquina que está ejecutando una tarea ya no puede hacerlo. Por tanto, puede migrar la tarea (checkpointing) hacia otra máquina que esté libre, continuando allí la ejecución, incluso si cambia el entorno de ejecución de Condor.

En aquellos casos donde Condor puede realizar checkpointing y migrar la tarea, Condor hace que resulte fácil el maximizar el número de máquinas que pueden ejecutar la tarea. En ese caso no hay requisitos para las máquinas de un sistema de ficheros distribuido (como por ejemplo NFS o AFS), por lo que las máquinas de toda una comunidad (un mismo Pool) pueden ejecutar una tarea, incluyendo las máquinas de diferentes dominios administrativos.

Condor puede ahorrar mucho tiempo, cuando una tarea debe de ejecutarse en muchas (cientos) de máquinas diferentes, con diferentes tipos de datos. Condor permite que con un solo comando se lancen todas las tareas. Dependiendo del número de máquinas que haya en el Pool, docenas o cientos de máquinas están preparadas para ejecutar la tarea en cualquier momento.

Condor no requiere tener una cuenta en las máquinas donde se ejecutan tareas. Condor puede hacer esto, gracias al sistema de llamada a procedimiento remoto (RPC) que toma las llamadas a las librerías para cada operación, como la lectura y escritura de ficheros. Estas llamadas son transmitidas a través de la red, para ser usadas por la máquina donde se ejecuta la tarea.

Condor implementa las ClassAds, un sencillo mecanismo que simplifica el lanzamiento de las tareas. Funciona de manera similar a los anuncios clasificados de un periódico. Todas las máquinas del Pool deben de indicar las propiedades de sus recursos, tanto los estáticos como los dinámicos. Un usuario especifica la solicitud de un recurso de ayuda cuando lanza una tarea. La fuente debe definir tanto los recursos necesitados, como los deseados para ejecutar tareas. Condor enlaza cada petición con su recurso correspondiente, para ejecutar la tarea. Durante este proceso Condor también considera los diferentes valores de prioridad.

Para terminar con esta introducción se explicarán algunas de las características especiales que posee Condor:

Migración y Checkpoint: cuando los programas pueden sen enlazados con las librerías de Condor, los usuarios de Condor pueden asegurarse de que las tareas sean completadas, incluso si se producen cambios en el entorno que Condor utiliza. Cuando

una máquina que ejecuta una tarea en Condor, se vuelve inutilizable, la tarea puede continuar, después de migrar a otra máquina.

No se necesitan hacer cambios en el código fuente de los usuarios: No se necesita una programación especial para usar Condor. Condor está preparado para ejecutar tareas no-interactivas. La migración y el checkpoint de los programas de Condor es transparente y automático, así como las llamadas a procedimiento remoto. Si se necesitan estas utilidades, el usuario sólo reenlaza las tareas. El código es recompilado, no cambiado.

Llamadas a procedimiento remoto: A pesar de ejecutar tareas en máquinas remotas, el modo de ejecución del universo Standard de Condor conserva el entorno de ejecución local a través de las llamadas a sistema remoto. Los usuarios no deben preocuparse de hacer que los ficheros de datos sean disponibles para las estaciones de trabajo remotas o también para obtener una cuenta de las estaciones remotas antes de que Condor ejecuten sus tareas allí. Los programas se comportan en Condor como si se estuvieran ejecutando en la estación de trabajo donde inicialmente se lanzaron sin importar en que máquina finalice realmente su ejecución.

Los Pools pueden trabajar de manera conjunta: Flocking es una característica de Condor que permiten que las tareas lanzadas en un Pool puedan ejecutarse en otro. El mecanismo es flexible, siguiendo las peticiones desde el lanzamiento de la tarea, mientras que son aceptadas por el segundo Pool, hasta el conjunto de máquinas de dicho Pool y se establecen las condiciones sobre las que las tareas se ejecutan.

Las tareas pueden ordenarse: El orden de ejecución necesario por las dependencias entre un conjunto de tareas puede establecerse de manera sencilla. El conjunto de las tareas emplea un grafo, donde cada nodo es una tarea. Las tareas son ejecutadas en Condor, siguiendo las dependencias dadas por el grafo.

Condor permite comportarse como un GRID: La técnica de glidein permite a las tareas lanzadas en Condor ser ejecutadas en máquinas grid en varias localizaciones alrededor del mundo.

Sensible a los deseos de los propietarios de las máquinas: El propietario de la máquina posee prioridad completa sobre el uso de la misma. El propietario generalmente permite la ejecución de las tareas de otros propietarios mientras que la máquina está en espera, pero desea que se termine su ejecución una vez que regrese. El usuario no debe hacer acciones especiales para recuperar el control. Condor realiza esta acción de manera automáticamente.

ClassAds: El mecanismo de ClassAd en Condor posee un marco muy flexible para unir las solicitudes de las fuentes con los recursos ofrecidos. Los usuarios pueden de manera sencilla solicitar los requisitos y los deseos de las tareas. El dueño de una estación de trabajo puede establecer una preferencia de modo que ejecute tareas de un conjunto específico de usuarios. El usuario también puede solicitar que no haya actividad entre las estaciones de trabajo a ciertas horas. Las preferencias, requisitos y las restricciones en la disponibilidad de los recursos, pueden se descritas en términos de expresiones potentes, permitiendo la adaptación de Condor de manera cercana, a cualquier política deseada.

2.2.3 Arquitectura y modos de funcionamiento

2.2.3.1 Funciones que una máquina puede desempeñar

Cada máquina en un Pool de Condor puede servir a una variedad de funciones. Muchas máquinas pueden desempeñar más de una función de manera simultánea. Ciertas funciones deben ser llevadas a cabo por máquinas únicas en el Pool. La

siguiente lista describe cuales son esas funciones, y que recursos con requeridos por la máquina que lleva a cabo el servicio:

Central Manager

Sólo puede haber un Central Manager en cada Pool. Esta máquina es la que recoge la información, y lleva a cabo la negociación entre las máquinas del Pool. Las dos responsabilidades del Central Manager son llevadas a cabo por demonios (procesos) separados, por lo que es posible tener máquinas diferentes dando los dos servicios. Sin embargo, normalmente se llevan a cabo por la misma máquina. Esta máquina toma una parte muy importante en el Pool de Condor, por lo que debe ser segura. Si esta máquina falla, no se podrán usar las utilidades de Condor. Por ello la máquina que sea el Central Manager debe mantenerse siempre activa, o en su defecto, debe ser aquella que pueda reiniciarse de la manera más rápida si algo sale mal. El Central Manager idealmente debe tener una buena conexión de red con todas las máquinas del Pool, ya que las máquinas deben de ser capaces de poner al día al Central Manager ante los posibles cambios. Todas las consultas van hacia el Central Manager.

Execute

Cualquier máquina del Pool (incluyendo el Central Manager) puede ser configurada para que pueda ejecutar o no tareas de Condor. De manera obvia, alguna de las máquinas debe realizar este papel, o el Pool no sería muy útil. Ser una máquina Execute no requiere mucho. La única característica es que tenga suficiente espacio de disco, lo suficiente para llevar a cabo la tarea, que será eliminada de la cola, una vez que haya mandado el resultado a la máquina que la lanzó. Sin embargo, si no se dispusiera de mucho espacio de disco, Condor simplemente limitaría el tamaño de las tareas que se pudieran lanzar. De manera general cuantos más recursos posea una máquina (memoria, velocidad de CPU..) más peticiones de las que le lleguen podrá servir. Sin embargo si son peticiones que no requieren muchas características, cualquier máquina del Pool podrá atenderlas.

Submit

Cualquier máquina del Pool (incluyendo el Central Manager) puede ser configurada, de manera que se la permita lanzar tareas de Condor. Las características de las peticiones para una máquina Submit van aumentando en comparación con los de las máquinas Execute. Lo primero de todo es que cualquier petición que se esté ejecutando en una máquina remota, genera otro proceso en la máquina Submit. Por ello si se tienen muchas tareas ejecutándose se necesita mucho espacio de memoria. Además todos los ficheros checkpoint de las tareas son almacenados en el espacio local que lanzó la tarea. Por lo tanto si la imagen de las tareas es muy grande, y se lanzan muchas de ellas, se necesitará mucho espacio de disco para almacenar esos ficheros. Los requisitos de espacio de disco pueden ser en cierto modo aliviados por un servidor Checkpoint, aunque los binarios de las tareas que se lanzan son guardados en la máquina Submit.

Servidor Checkpoint

Una máquina del Pool puede configurada como un Servidor Chekpoint. Esto es opcional, y no es parte de la distribución binaria estándar de Condor. El Servidor Checkpoint es una máquina centralizada que almacena todos los ficheros checkpoint de las tareas que se lanzan en el Pool.

Esta máquina debe poseer mucho espacio de disco y una buena conexión de red hacia el resto del Pool, puesto que el tráfico puede ser denso.

2.2.3.2 Procesos (Demonios) de Condor

La siguiente lista describe todos los demonios y programas que funcionan en Condor y la tarea que desempeñan:

condor_master

Este demonio es el responsable de mantener el resto de los demonios ejecutándose en cada máquina del Pool.

Genera el resto de los demonios, y periódicamente comprueba si hay nuevos binarios instalados para cada uno de ellos. Si los hay condor_master reiniciará el resto de los demonios. Además si algún demonio el demonio falla, el master mandará un correo al administrador del Pool de Condor y reiniciará el demonio. El master también soporta varios comandos administrativos que permiten iniciar, parar o reconfigurar los demonios de manera remota. A continuación se explican los más utilizados:

condor_on	Inicia parte de los demonios en las máquinas que se le indiquen. Este comando asume que el master se está ejecutando en cada máquina. Si no es el caso no podrá encontrar la dirección del master. Los demonios que se inicien son los que aparecen en la lista DAEMON_LIST del archivo de configuración
condor_off	Apaga los demonios de la lista DAEMON_LIST de las máquinas que se la indique. Esto lo hace de manera limpia por los que las tareas del tipo checkpoint puden terminar de manera exitosa con la mínima pérdida de trabajo.
condor_reconfig	Reconfigura todos los demonios de Condor en concordancia con el estado actual de los archivos de configuración. Hay algunas características de la configuración que sólo pueden renovarse si se usa este comando.
condor_restart	Reinicia los demonios de una serie de máquinas. El demonio será puesto en un estado consistente, eliminado y comenzado de nuevo.
condor_config_val	Sirve para poder ver la instalación actual de Condor en una máquina dada. Dada una lista de variables Condor indicará cual de esas variables está activada. También puede ser usada para cambiar el valor de determinadas características de los demonios de Condor. Ningún cambio tendrá efecto a menos que se ejecute condor_reconfig.

Tabla 2. 1 Comandos útiles del demonio master

condor_master se ejecutará en cada máquina del Pool, independientemente de la función que desempeñe cada máquina. Con cierta periocidad, se ejecutará el comando condor_preen para eliminar aquellos ficheros que ya no le sean necesarios a Condor.

condor_startd

Este demonio representa un recurso (máquina capaz de ejecutar tareas) dado del Pool. Advierte de ciertos atributos que debe tener la máquina Execute, dependiendo de las características de las tareas que se lanzan. Este demonio se ejecutará en cada máquina del Pool a la que se le permita ejecutar tareas. Es responsabilidad de los que lanzar las tareas determinar bajo que condiciones las tareas deben ser empezadas, suspendidas, etc. Cuando el demonio startd está preparado para ejecutar una tarea de Condor genera el demonio condor_starter.

También posee ciertos comandos administrativos. Uno de los más importantes es condor_vacate lo que permite a Condor realizar checkpoint ejecutándose en una serie de máquinas y permite a las tareas abandonar la máquina, aunque el trabajo permanecerá en la cola hasta que sea definitivamente eliminado. Si se posee máquinas con multiprocesadores, se tendrá un demonio condor_startd por procesador.

condor_starter

Este demonio es la entidad que actualmente genera la tarea de Condor en una la máquina remota. Inicia el entorno de ejecución y monitoriza la tarea una vez que se está ejecutando. Cuando se termina la tarea, el demonio starter lo notifica mandado la información de estado necesaria a la máquina Submit, y termina la ejecución.

condor_schedd

Este demonio representa la solicitud de recursos al Pool. Cualquier máquina que se desee que pueda lanzar tareas, necesitan tener el demonio condor_schedd ejecutándose.

Cuando un usuario lanza una tarea, esta se remite al demonio schedd, que la sitúa en la cola de tareas, la cual controla. Muchas herramientas que permiten manejar y visualizar la cola, deben remitirse a este demonio para realizar su trabajo :

condor_submit	Es el comando que se encarga para lanzar las tareas a través de Condor. Para ello se requiere un fichero de descripción submit que se comentará más adelante
condor_rm	Elimina las tareas de la cola
condor_q	Muestra las tareas que se encuentran en la cola.

Tabla 2. 2 Comandos útiles del demonio schedd

Si el demonio schedd se para, ninguna de estas funciones podrá llevarse a cabo. El demonio schedd controla el número de tareas en espera en la cola de trabajo, y es responsable de encontrar máquinas adecuadas para realizar las peticiones. Una vez que el schedd se ha asignado a un recurso, el schedd genera el demonio condor_shadow, que servirá a esa petición en particular.

condor_shadow

Este demonio se ejecuta en la máquina donde se lanzó la tarea y actúa como el manejador de los recursos para dicha petición. Las tareas que son enlazadas hacia el Universo Standard (el cual se explicará en la siguiente sección), realizan llamadas a procedimientos remotos a través del condor_shadow. Una llamada a sistema creada en la máquina Execute remota se envía a través de la red, hacia el condor_shadow que realizará la llamada del sistema (como las operaciones de entrada/salida) en la máquina que lanzó la tarea, y el resultado se envía de nuevo a través de la red a la tarea remota.

Además, este demonio es el responsable de tomar decisiones sobre las peticiones (como por ejemplo donde deben almacenarse los ficheros checkpoint, como debe de ser el acceso a ciertos archivos, etc.). Uno de los comandos más importantes que emplea es condor_status. Es una herramienta versátil que puede ser empleada para monitorizar y controlar el Pool. Puede usarse para consultar información de los recursos

condor_collector

Este demonio es el responsable de recolectar la información de estado de todo el Pool de Condor. El resto de los demonios, de manera periódica enviarán ClassAd para poner al día al collector. Estos ClassAds contienen información completa sobre el estado de los demonios, los recursos que representan o los recursos necesarios para las tareas del Pool (como tareas que deben de tener un determinado schedd). El comando condor_status puede ser usado para preguntar al collector acerca de información específica sobre varias partes de Condor. Asimismo los demonios de

Condor, por sí mismos preguntan al collector sobre información importante, como por ejemplo que dirección usar para enviar comandos a una máquina remota.

condor_negotiator

Este es el demonio responsable de todas las negociaciones que se llevan a cabo en el Pool. De manera periódica, este demonio comienza un ciclo de negociación, donde pregunta al collector sobre el estado actual de todos los recursos del Pool. Contacta con cada schedd que está esperando para satisfacer las peticiones en orden de prioridad, y trata de encontrar los recursos adecuados para satisfacer dichas peticiones. Este demonio es el responsable de hacer cumplir las prioridades del sistemas, donde cuantas más características un determinado usuario ha solicitado, se le otorgará menor prioridad en las próximas peticiones. Si un usuario con una mayor prioridad tiene tareas que están esperando para ejecutarse, y un usuario con una prioridad menor solicita recursos, el demonio negotiator puede adelantar dicho recurso y hacerlo de manera paralela a atender la petición del usuario con la mayor prioridad.

Nota: La prioridad de un usuario será mayor cuanto menor sea el valor numérico que tenga dicha prioridad (es decir, una prioridad de 0.5 es mejor que una prioridad de 1).

condor_kbdd

Este demonio solo se emplea en Digital Unix. En dichas plataformas condor_startd no es capaz de determinar la actividad de consola (como el ratón o el teclado) de manera directa por el sistema. El demonio kbdd conecta con el servidor X y comprueba de manera periódica si ha habido alguna actividad. Si la hay kbdd manda un comando al startd. De esa manera el demonio startd reconoce que el usuario está usando la máquina de nuevo y puede decidir que acciones son necesarias.

condor_ckpt_server

Éste es el servidor checkpoint. Da servicio a las tareas de almacenamiento y envío de ficheros checkpoint. Si el Pool ha sido configurado para usar un servidor checkpoint, pero esa máquina cae Condor vuelve a mandar los ficheros checkpoint de una determinada tarea de vuelta a la máquina que lanzó la tarea.

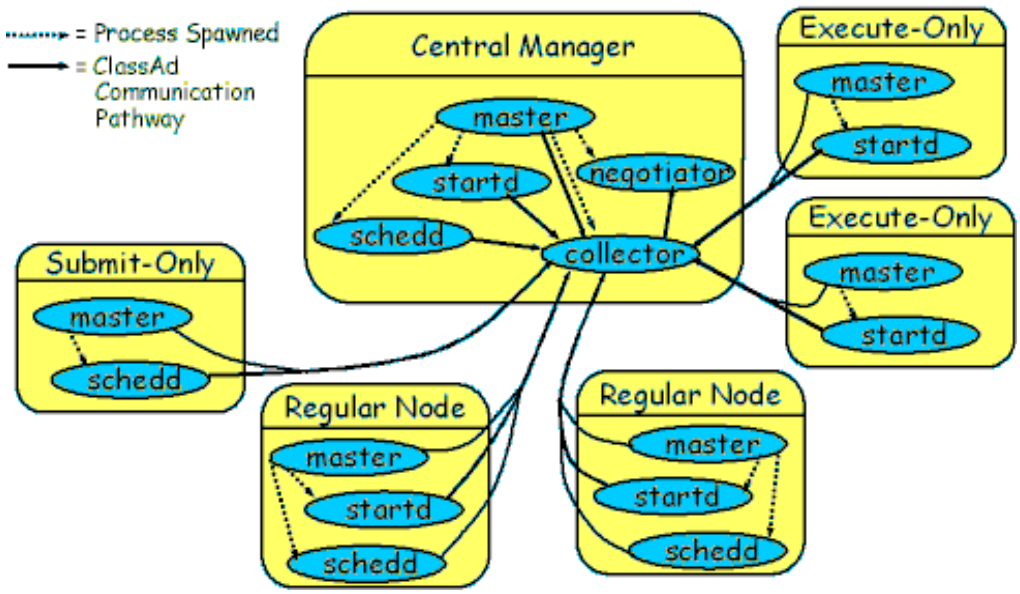


Figura 2. 1 Demonios de un Pool

2.2.3.3 Estados de las máquinas de un Pool.

Condor le asigna un estado a cada máquina. Este estado depende de si a la máquina se le permite lanzar tareas, y también de en que parte de la negociación se encuentre. Los posibles estados son:

Owner

El estado Owner representa a una máquina que está siendo usada por el propietario de la misma. La máquina permanece en este estado, mientras que la variable isOwner es true. Si la expresión isOwner se evalúa a false la máquina pasa al estado Unclaimed. El valor por defecto de esta expresión se selecciona dependiendo de cada Pool. La máquina permanecerá en el estado Owner tanto como la expresión START permanezca a false. Si la expresión START pone a true, o no se define, la transición de Owner a Unclaimed se produce.

Cuando se tienen máquinas dedicadas para el Pool, es decir, máquinas que sólo se encarguen de ejecutar tareas de Condor, se recomienda que la expresión isOwner se le asigne el valor false. El estado Unclaimed representa una máquina que no está siendo usada ni por el propietario de la misma, ni por el sistema Condor. Desde el punto de vista de Condor, apenas hay diferencia entre ambos estados. En ambos casos el recurso no está siendo usado por Condor. Sin embargo si una tarea solicita un recurso, la máquina está preparada para ejecutar la tarea, independientemente de que la máquina se encuentre en uno u otro estado.

La única diferencia entre uno u otro estado es de cómo el recurso es percibido por el demonio condor_status y otras herramientas de evaluación, y el hecho es que Condor no ejecutará benchmarking en un recurso que se encuentre en el estado Owner. Tan pronto como la expresión IsOwner se ponga a true la máquina volverá al estado Owner. Cuando la expresión se ponga a false, entrará en el estado Unclaimed. Un ejemplo de la expresión START es el siguiente:

START =KeyBoardIdle >15*$(MINUTE) || Owner == “Condor”

Esta expresión indica que cuando la máquina permanezca sin usar el teclado durante 15 minutos la máquina pasará del estado Owner al de Unclaimed. La expresión || indica que la máquina, aún estando en el estado Owner, si tiene tareas del usuario “Condor” tiene la posibilidad de llevarlas a cabo.

Mientras que la maquina está en el estado Owner, el demonio condor_startd comprobará el estado de la misma cada *UPDATE_INTERVAL*, para comprobar si algo ha cambiado, y la máquina debe ser llevada hacia otro estado. Esto minimiza el impacto sobre el usuario mientras que este está usando la máquina. La máquina sólo puede llegar al estado Unclaimed a través del estado Owner.

Unclaimed

Si la expresión isOwner se evalúa a true, entonces la máquina regresa al estado Owner. Si la expresión isOwner es false, entonces la máquina permanece en el estado Unclaimed. Si la expresión isOwner no aparece en los archivos de configuración, entonces el valor por defecto de esta expresión es:

START=?=FALSE

Por ello mientras que se encuentre en el estado Unclaimed, si la expresión START local se evalúa a false, la máquina regresa al estado Owner.

Cuando se encuentra en el estado Unclaimed, la expresión RunBenchmarks es importante. Si la expresión RunBenchmarks es true mientras que la máquina está en el estado Unclaimed, entonces la máquina pasa de la actividad Idle, a la actividad Benchmarking, y ejecuta benchmarks para calcular MIPS y KFLOPS. Cuando se completan las benchmarks, la máquina regresa al estado Idle. El demonio

condor_startd automáticamente inserta un atributo, LastBenchmark, cada vez que se ejecute benchmarks, por ello comúnmente Runbenchmarks se define en términos de de este atributo, por ejemplo:

```
BenchmarkTimer=(CurrentTime-LastBenchmark)
RunBenchmarks=$(BenchmarkTimer)>=(4*$(HOUR))
```

La expresión BenchmarkTimer calcula el tiempo que transcurre desde la última benchmark, por lo que cuando este tiempo execede de las 4 horas, se ejecutan benchmarks de nuevo. El demonio condor_startd guarda una media de estos tiempos, para seleccionar los recursos con los mejores tiempos

Matched

A la máquina se le permite ejecutar tareas, y ha sido escogido por el demonio negotiator con un schedd específico. Este schedd no ha reclamado aún esta máquina. En ese estado a la máquina no se le permite atender a más tareas. El estado Matched no es muy interesante para Condor. La máquina miente sobre su estado en la expresión START mientras que está en este estado, y establece la variable Requeriments a false para prevenir que sea seleccionada otra vez antes de que sea Claimed. Algo también interesante, es que el demonio condor_startd, empieza un temporizador, para asegurarse de que no permanece en este estado demasiado tiempo. Este temporizador se establece en la expresión MATCH_TIMEOUT del fichero de configuración. Se establece en segundos. Si el schedd al que se le asignó esta máquina, no la reclama después de este período de tiempo, la máquina regresa al estado Owner.

Claimed

La máquina ha sido solicitada por un schedd. El estado claimed es el más complejo de todos los estados. Tiene el mayor número posible de actividades, y el mayor número de expresiones que determinan sus próximas actividades. Además los comandos condor_checkpoint y condor_vacate afectan a la máquina cuando se encuentra en el estado Claimed. En general hay dos grupos de expresiones del archivo de configuración global que tienen efecto sobre este estado, dependiendo del tipo de Universo seleccionado. Por ejemplo:

```
WANT_SUSPEND=True
WANT_VACATE=$(ActivationTimer) > 10 * $(MINUTE)
SUSPEND=$(KeyboardBusy) || (CPUBusy)
```

Si se encontrase en el caso del Universo Vanilla, como es el caso de este proyecto, las expresiones poseen la cadena “_VANILLA” detrás de los nombres. Por ejemplo:

```
WANT_SUSPEND_VANILLA=True
WANT_VACATE_VANILLA=True
SUSPEND_VANILLA=$(KeyboardBusy) || (CPUBusy)
```

Si no se especifica el Universo Vanilla, Condor tomará el valor por defecto, que es el primero. El usuario debe especificarlo, ya que la principal diferencia estará en que las máquinas no realizaran checkpointing. Por ejemplo, un usuario podría desear, que las tareas del Universo Vanilla permanecieran más tiempo en espera, que las del Universo Standard. Mientras que la máquina se encuentra en el Universo Claimed, la variable POLLING_INTERVAL empieza a funcionar, y el demonio condor_startd, chequea la máquina más frecuentemente, para evaluar su estado. Si el propietario de la máquina empieza a usar de nuevo la consola, lo mejor es que Condor lo sepa lo antes posible, para ser capaz de empezar a hacer lo que el propietario de la máquina desee en dicho momento.

Hay una gran variedad de eventos, que pueden causar que el demonio condor_startd trate de deshacerse, o suspender de manera temporal un trabajo en ejecución. La actividad de consola de la máquina, descarga de archivos desde otras tareas, o el apagado del demonio condor_startd, por medio de un comando administrativo, pueden ser posibles causas de interferencias. Otra causa puede ser la aparición de una tarea con mayor prioridad, que quiera ejecutarse en la máquina. Dependiendo de la configuración, el demonio condor_startd puede actuar de una manera un poco diferente dependiendo de la máquina. El demonio condor_startd puede ser configurado para ignorar completamente la actividad de la consola, o para suspender la tarea, o también para eliminarla.

La expresión WANT_SUSPEND determina si la máquina debe evaluar la expresión SUSPEND, para considerar la completa suspensión de la tarea. La expresión WANT_VACATE, determina que ocurre cuando la máquina entra en el estado Premmpting. Este estado puede ir desde la suspensión, a la completa eliminación de la tarea. Si una o más de estas expresiones se pone a false, la máquina saltará de este estado eliminando la tarea, y procediendo directamente con los cambios más drásticos.

Si un comando condor_checkpoint se ejecuta, o la expresión PeriodicCheckpoint se evalúa a true, no hay ningún cambio de estado. El demonio condor_startd no tiene ninguna manera de conocer cuando se completa este proceso, por ello de manera periódica comprueba que no hay cambios.

Preempting

Este estado es menos complejo que el estado Claimed. Hay dos actividades. Dependiendo del valor de la expresión WANT_VACATE, una máquina se encuentra en la actividad Vacating (si está a True), o en la actividad Killing (si está a False). Mientras que se está en el estado Preempting, la máquina comprueba si la expresión Requirements es False, lo que significa que no está disponible para máquinas ajenas, por diferentes causas:

1) El propietario de la máquina la está utilizando
2) Otro usuario con una prioridad mayor tiene tareas esperando para ejecutarse.
3) Se ha encontrado otra solicitud que esta máquina debe servir antes.

Si la máquina se encuentra en la actividad Killing (ya sea porque la expresión WANT_VACATE es False, o porque la expresión KILL es True), se fuerza al condor_starter, para eliminar de manera inmediata la tarea. Una vez que la máquina ha comenzado a eliminar de manera forzosa la tarea, el demonio condor_startd comienza un temporizador, la duración del cual queda definido por la expresión KILLING_TIMEOUT (por defecto 30 segundos). Si este temporizador expira, la máquina actúa en la actividad Killing, algo puede ir realmente mal con el demonio condor_starter, y el condor_startd trata de eliminar la tarea, mediante el envío de la señal SIGKILL al condor_starter.

Una vez que el condor_starter ha eliminado todos los procesos asociados con la tarea, y una vez que se notifica al condor_schedd que la demanda no ha sido satisfecha, el estado de Preempting se abandona.

Backfill

Cuando una máquina está esperando a que vuelva el dueño de la máquina o a que se le asigne una tarea de Condor. Este estado sólo se produce si la máquina está especialmente configurada para admitir tareas backfill.

2.2.3.4 Actividades en los estados de una máquina.

Las actividades que se pueden tener dependiendo de los estados se enumeran a continuación:

Owner

Idle	Tan pronto como Condor se da cuenta que la máquina está en el estado Idle, Condor sabe que esta máquina no va a realizar ninguna actividad para Condor.

Unclaimed

Idle	La máquina permanece en este estado hasta que el usuario deja la máquina.
Benchmarking:	La máquina está ejecutando benchmarks para determinar la velocidad de la máquina. Esta actividad sólo ocurre en el estado Unclaimed. Cada cuanto realiza esta actividad se determina por la expresión *RunBencmarks*.

Matched

Idle	La máquina permanece en este estado hasta que el usuario la deja.

Claimed

Idle	la máquina ha sido reclamada por Condor, pero el schedd que la reclamó aún no ha llamado al condor_starter para que de servicio a la tarea. La máquina regresa a este estado (normalmente de manera breve) cuando las tareas (y por consiguiente el condor_starter) acaban..
Busy:	Una vez que el condor_starter empieza y el estado Claimed se activa, la máquina se mueve a la actividad Busy para indicar que está haciendo algo la máquina que le interesa a Condor.
Suspended:	Si el trabajo ha sido suspendido por Condor, la máquina entra dentro de la actividad Suspended. La relación entre la máquina y el schedd no se ha roto, pero el trabajo no está realizando ningún progreso y Condor no va a generar ninguna actividad en la máquina.
Retiring	Cuando una actividad es reclamada para ser cancelada por alguna razón mientras que espera a que se termine la tarea entra en este estado. La expresión MaxJobRetirementTime determina cuanto tiempo debe esperar. Una vez que el trabajo finaliza o el tiempo d espera se acaba, se entra en el estado Preempting.

Preeempting

Vacating:	En esta actividad la tarea que está ejecutándose está en el proceso de checkpointing. Tan pronto como el proceso de checkpointing se ha completado, el trabajo entra en el estado Owner o en el estado Claimed, dependiendo del motivo por el que se canceló.
Killing:	L la máquina ha solicitado que se elimine la tarea de manera inmediata, sin realizar checkpointing.

Backfill

Idle	La máquina ha sido configurada para que ejecute tareas backfill y está preparada para hacerlo, pero no ha tenido tiempo para crear un controlador backfill.
Busy	La máquina está realizando una ejecución backfill..

Killing	La máquina estaba realizando una ejecución backfill, pero ahora mismo está eliminando la tarea, bien porque el usuario lo ha solicitado, o para realizar una tarea normal de Condor..

2.2.3.5 Relaciones Estado-Actividad.

La siguiente figura trata de mostrar de manera gráfica las relaciones entre los diferentes estados, y las actividades que se pueden llevar a cabo en cada uno de ellos. Cada una de las flechas indica una transición de un estado a otro indicándose mediante colores el estado del que proceden, y con una letra el orden en el que se producen estas transiciones.

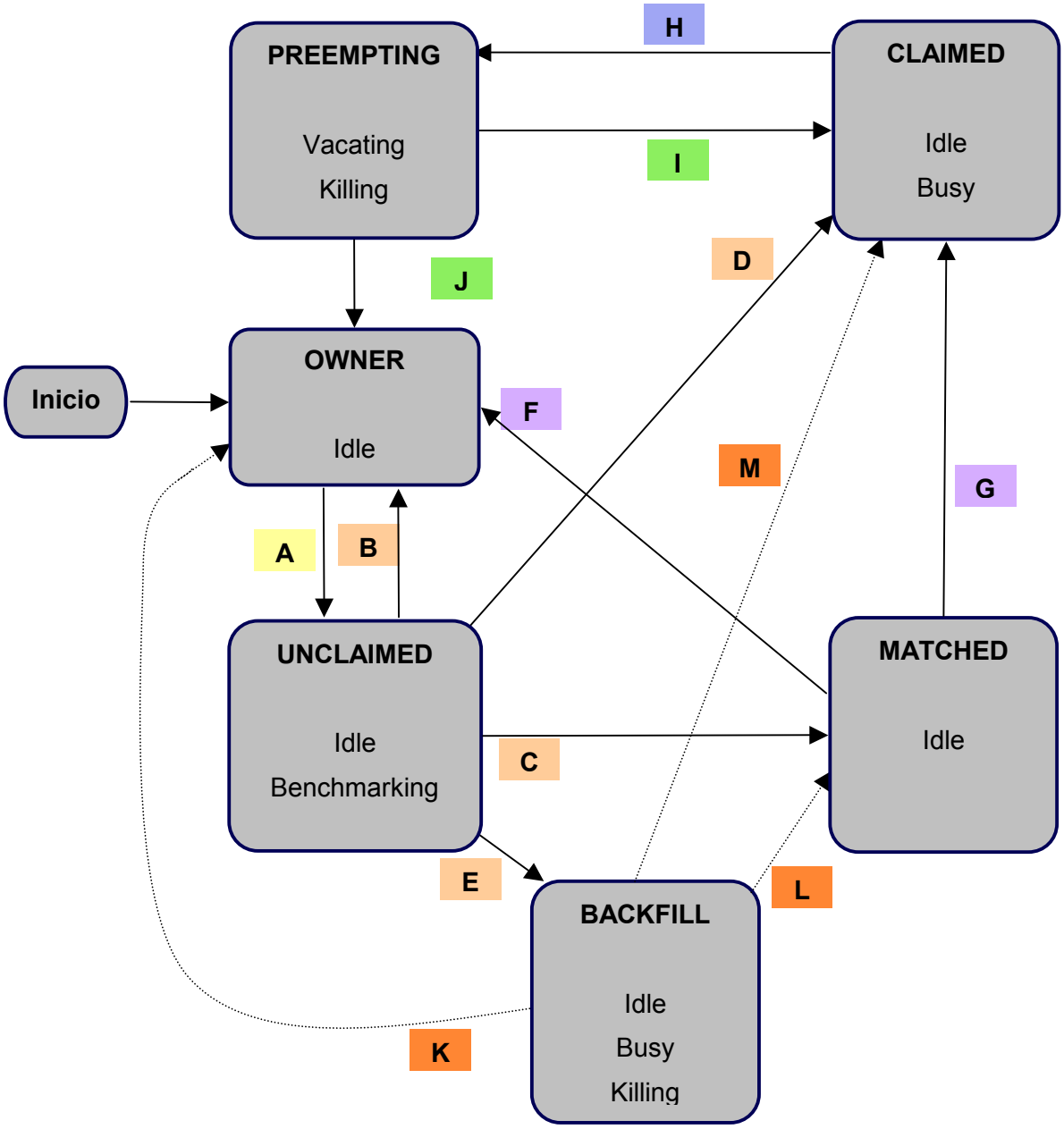


Figura 2. 2 Posibles estados y actividades de un Pool de Condor

- ***Transiciones desde el estado Owner***

  A. La máquina pasa del estado Owner al de Unclaimed, cada vez que la expresión START (del archivo de configuración) cambia a trae. Esto indica que la máquina está capacitada para ejecutar una tarea de Condor.

- ***Transiciones desde el estado Unclaimed***

  B. La máquina vuelve del estado Unclaimed al de Owner cada vez que la expresión START vuelve a tener el valor de false, eso significa que la máquina no está preparada para ejecutar tareas de Condor, porque está siendo usada por el propietario de la máquina.

  C. La transición desde el estado Unclaimed al de Matched ocurre cada vez que el demonio **condor_negotiator** le asigna una tarea.

  D. La transición desde el estado Unclaimed directamente al estado Claimed, también ocurre si el **condor_negotiator** le asigna una tarea. En este caso el **condor_schedd** recibe la asignación e inicia el protocolo claiming con la máquina, antes de que el **condor_startd** reciba la asignación de la tarea desde el **condor_negotiator**.

  E. La transición desde el estado Unclaimed al de Backfill ocurre si la máquina está configurada para ejecutar tareas Backfill, y la expresión START_BACKFILL se evalúa a trae.

- ***Transiciones desde el estado Matched***

  F. La máquina se mueve desde el estado Matched al de Owner si la expresión START vuelve a tener el valor de FALSE, o cada vez que la expresión MATCH_TIMEOUT expira. Este timeout se usa para asegurarse que si a una máquina el **condor_schedd** le asigna una tarea, pero ese **condor_schedd** no consigue contactar con el **condor_start** para llevarla a cabo, la máquina queda liberada de dicha petición, y está preparada para la siguiente asignación. En ese caso como la expresión START no ha sido evaluada a false, tan pronto como la transición **F** se completa, la máquina entrará inmediatamente en el estado Unclaimed de nuevo (a través de la transición **A**). La máquina también irá del estado Matched al de Owner si el **condor_schedd** trata de llevar a cabo el protocolo claiming pero ocurre alguna clase de error. Finamente la máquina si el demonio **condor_startd** recibe un comando **condor_vacate** mientras que se encuentra en el estado Matched.

  G. La transición desde el estado Matched al de Claimed ocurre cuando el **condor_schedd** completa de manera satisfactoria el protocolo claiming con el demonio **condor_startd**.

- ***Transiciones desde el estado Claimed***

  H. Desde el estado Claimed, el único destino posible es el estado Preempting. Esta transición puede ocurrir por diversas razones:

    - El demonio **condor_schedd** que ha reclamado la máquina no tiene más tareas que ofrecer.
    - La expresión PREEMPT se evalúa a true (lo que normalmente significa que el dueño dela máquina está haciendo uso de la misma).
    - El demonio **condor_startd** recibe el comando **condor_vacate**.
    - El demonio **condor_startd** se apaga (ya sea por una señal o por el comando **condor_off**).

- A la máquina se le asigna una tarea con una prioridad mayor (ya sea un usuario con una mayor prioridad, o que la expresión RANK de la máquina sea mejor).

➢ ***Transiciones desde el estado Preempting***

I. La máquina se moverá desde el estado Preeempting de regreso al de Claimed si se le asigna una tarea con una prioridad mayor.

J. La máquina se moverá del estado Preempting al de Owner si la expresión PREEMPT se evalúa a true, si se usa el comando **condor_vacate**, o si la expresión START se evalúa a false cuando el demonio **condor_startd** termina eliminando cualquier tarea que se estuviera ejecutando cuando entró en el estado Preempting

➢ ***Transiciones desde el estado Backfill***

K. La máquina se moverá del estado Backfill al de Owner por las siguientes razones:

- La expresión EVICT_BACKFILL se evalúa a true.
- El demonio **condor_startd** recibe el comando **condor_vacate**.
- El demonio **condor_startd** de apaga.

L. La transición desde el estado Backfill al de Matched ocurre cada vez que una máquina que ejecuta una tarea backfill es asignada por un demonio **condor_schedd** que desea ejecutar una tarea de Condor.

M. La transición desde el estado Backfill directamente hacia el estado Claimed es similar a la transición desde el estado Unclaimed directamente al estado Claimed. Sólo ocurre si el demonio **condor_schedd** completa el protocolo claiming antes de que el demonio **condor_schedd** reciba la notificación de la asignación de la tarea desde el demonio **condor_negotiator**.

2.2.3.6 Expresiones influyentes en las transiciones Estado/Actividad

Por último, y para terminar con este apartado sobre los estados, sus actividades y las transiciones entre los mismos, se va a realizar un resumen sobre las expresiones más influyentes, en los diferentes estados, de los archivos de configuración.

START: Cuando es True, la máquina está preparada para ejecutar tareas remotas.

Runbenchmarks: Mientras que se encuentra en el estado Unclaimed, la máquina deberá de ejecutar benchmarks cada vez que su valor sea true. En este proyecto ha sido necesario establecer el valor de esta variable a false, porque produce graves problemas a la hora de ejecutar tareas.

MATCH_TIMEOUT: Si la máquina ha permanecido en el estado Unclaimed más tiempo del que establece esta expresión, se producirá una transición al estado Owner.

WANT_SUSPEND: Si su valor es true, la maquina comprueba el valor de la expresión SUSPEND, para comprobar si debe suspender la tarea. Si su valor es false, comprobará la expresión PREEMPT.

SUSPEND: Si la expresión WANT_SUSPEND es true, y la máquina está en el estado Claimed, suspende la tarea si el valor de la variable SUSPEND es true.

CONTINUE: Si la máquina está en el estado Claimed, entra en la actividad Busy si la expresión CONTINUE es true.

PREEMPT: Si la máquina está en el estado Claimed, ejerciendo la actividad de Suspended o Busy, y la expresión WANT_SUSPEND es false, la máquina pasa a la actividad Retiring, si el valor de la variable preempt es true.

CLAIM_WORKLIFE: Si se especifica, esta expresión especifica el número de segundos durante los cuales un recurso continuará aceptando peticiones. Una vez que este temporizador expira, cualquier tarea existente, continuará ejecutándose de manera normal, pero una vez que la tarea termina, o queda en el estado preempting, el recurso deja de aceptar o ejecutar tareas. Esto es útil si se desea renegociar de manera periódica con las máquinas.

MAXJOBRETIREMENTTIME: Si la máquina se encuentra en el estado Claimed, en la actividad de Retiring, esta expresión especifica el número máximo de segundos, que el demonio condor_startd a la tarea, para terminar de manera natural (sin señales que fuercen su eliminación). El reloj se inicia cuando la tarea comienza y se para durante cualquier suspensión. La tarea debe ofrecer su propio MaxjobRetirementTime, pero sólo puede ser usada para que este tiempo sea menor que el proporcionado por el demonio condor_startd, nunca más. Una vez que la tarea acaba, o este tiempo expira, la máquina entra en el estado Preempting.

WANT_VACATE: Esta expresión solo se comprueba cuando la expresión PREEMPT es true y la máquina entra en el estado Preempting. Si WANT_VACATE es true, la máquina entra en la actividad Vacating. Si su valor es false, la máquina procederá directamente con la actividad Killing.

KILL: Si la máquina está en el estado Preempting, en la actividad de Vacating, entra en el estado Preempting, en la actividad de Killing, cuando el valor de esta variable sea true.

KILLING_TIMEOUT: Si la máquina está en el estado de Preempting, durante la actividad de Killing, y expira este contador, el demonio manda una señal SIGKILL al demonio condor_starter, para eliminar la tarea lo más pronto posible.

PERIODIC_CHECKPOINT: Si la máquina está en el estado Claimed, en la actividad Busy, y esta variable es true, la tarea ejecutará un checkpoint periódico

RANK: Si esta expresión se evalúa a un valor mayor por parte de una petición pendiente, que la que se está ejecutando en la máquina, la máquina pasa al estado Preempt la petición actual. Cuando se completa este estado, la máquina entra en el estado Claimed con la nueva petición.

START_BACKFILL: Cuando se encuentra a true, si la máquina está en la actividad de Idle, entrará en el estado Backfill, y realizará tareas Backfill (usando BOINC).

EVICT_BACKFILL

2.2.4 Universos

Los Universos de Condor definen los entornos de computación. El Universo debe especificarse en el fichero de descripción submit. Si no se especifica ningún Universo, Condor toma por defecto el Universo Standard. Los Universos soportados por esta versión son:

2.2.4.1 Universo Standard

En el Universo Standard Condor permite llamadas a procedimiento remoto y checkpointing. Estas características hacen que las tareas se ejecuten más fácilmente, y permite acceso remoto desde las fuentes hacia cualquier lugar en el Pool. Para preparar una tarea, como una tarea del Universo Standard, debe ser enlazada con condor_compile. Muchos programas pueden prepararse como tareas del Universo Standard, pero hay algunas restricciones.

Condor realiza el checkpoint de una tarea a intervalos regulares. Una imagen checkpoint es básicamente una captura del estado actual de una tarea. Si una tarea debe moverse de una máquina a otra, Condor realiza una imagen de la tarea, copia dicha imagen en la nueva máquina, y reinicia la tarea continuando por donde se había quedado. Si una máquina suele pararse o reiniciarse cuando ejecuta tareas, Condor puede reiniciar la tarea en una nueva máquina usando la imagen checkpoint más reciente que conserve. Gracias a esto las tareas pueden durar meses o años aunque la máquina ocasionalmente falle.

Las llamadas a procedimiento remoto se hacen de manera transparente al usuario, es decir, el usuario la percibe como si se estuvieran ejecutando en su propia máquina. Cuando una tarea se ejecuta en una máquina remota, un segundo proceso llamado condor_shadow corre en la máquina donde se lanzó la tarea.

Cuando la tarea atiende una llamada de sistema, condor_shadow realiza dicha llamada y manda los resultados a la máquina remota. Por ejemplo, si una tarea (en la máquina remota) necesita un fichero situado en la máquina que lanzó la tarea, condor_shadow buscará la tarea y enviará la información a la máquina donde se ejecuta la tarea.

Para convertir los programas en tareas del Universo Standard, se debe usar condor_compile para unir el programa a su correspondiente librería. Cuando se usa condor_compile no es necesario modificar el código fuente, pero se necesita tener acceso a los fichero *object*. Un programa comercial que está empaquetado en un solo fichero ejecutable no puede ser ejecutada por el Universo Standard. Por ejemplo si se desea enlazar el trabajo ejecutando

```
%    cc main.o tools.o -o pogram
```

entonces deberá relanzarse el trabajo para Condor con:

```
% condor_compile cc main.o tools.o –o program
```

Hay algunas restricciones sobre las tareas que pueden lanzarse en el Universo Standard:

1. Las tareas multi-procesador no están permitidas, esto incluye llamadas de sistema como fork(), exec(), and system().
2. No se permite la comunicación entre procesos.
3. Las comunicaciones en red deben ser cortas. –una tarea debe hacer conexiones de red usando llamadas de sistema, como son los sockets, pero las comunicaciones que permanezcan abiertas mucho tiempo retrasarán la migración y el checkpointing.
4. Enviar o recibir señales SIGUSR2 o SIGTSTP no está permitido. Condor reserva esas señales para su propio uso. Enviar o recibir otras señales si está permitido.
5. Alarmas, temporizadores, y señales de apagado no están permitidas. Esto incluye alarm(), getimer(), y sleep().

6. Los hilos múltiples a nivel de kernel no se permiten. Se permiten a nivel de usuario.
7. Los archivos de mapas de memoria no se permiten. Esto incluye las llamadas del tipo nmap() y munmap().
8. Los ficheros lock, pero no se guardan entre checkpoints.
9. Todos los ficheros deben abrirse como si fueran de “sólo lectura” o sólo escritura. Si los ficheros fueran de los dos tipos darían problemas a la hora de hacer las tareas de migración.
10. Si se realizan tareas de checkpoint se debe reservar una buena cantidad de espacio en disco, para almacenar las imágenes de las tareas que no han finalizado. Si el espacio de disco que se tiene es pequeño se puede designar una máquina para que haga de servidor Checkpoint, que almacenará todas las imágenes de las tareas.
11. En Linux las tareas deben ser enlazadas de manera estática. El enlace dinámico está permitido en las demás plataformas.

2.2.4.2 Universo Vanilla

El Universo Vanilla está diseñado para programas, que no pueden ser reenlazadas de manera satisfactoria. Los shell scripts son otro caso donde el Universo Vanilla es útil. Esto tiene consecuencias desafortunadas para las tareas que no son completadas en la máquina remota antes de que sean devueltas a su dueño. En este caso Condor sólo tiene dos opciones. Se pueden suspender las tareas, esperando a que se finalicen a posteriori, o puede comenzar el trabajo desde el principio en otra máquina del Pool. Como Condor no puede realizar llamadas a procedimiento remoto en el Universo Vanilla, el acceso a las entradas o salidas de las tareas se convierte en un problema. Una opción para Condor es depender de un sistema de ficheros distribuido como NFS o AFS. De manera alternativa, Condor tiene un mecanismo para transferir archivos en beneficio del usuario. En ese caso Condor transferirá todos los ficheros necesarios para realizar una tarea al lugar de ejecución, ejecutar la tarea, y transferir la salida de nuevo a la máquina que la lanzó.

En Unix Condor supone que se usa un sistema de ficheros distribuido para las tareas del Universo Vanilla. Sin embargo si no se dispone de dicho sistema un usuario puede activar el mecanismo de transmisión de ficheros de Condor. En las plataformas Windows, este mecanismo se usa por defecto.

2.2.4.3 Universo PVM

Este Universo permite a los programas escritos por la interfaz Parallel Virtual Machine ser usadas a través del entorno correspondiente de Condor.

2.2.4.4 Universo Grid

Es necesario proveer la interfaz de Condor a los usuarios que deseen lanzar tareas en sistemas administrados de manera remota.

2.2.4.5 Universo Java

Un programa del Universo Java debe poder ejecutarse en cualquier clase de máquina a pesar de su localización, propietario, o su versión JVM. Condor tendrá cuidado de todos los detalles, como buscar los binarios de JVM y establecer el classpath.

2.2.4.6 Universo Scheluder

Permite a los usuarios lanzar tareas pequeñas, para que sean ejecutadas de manera inmediata, a través del demonio **condor_schedd** por el mismo host Submit. Las tareas del Universo Scheluder no son emparejadas con una máquina remota, y nuncan pueden ser adelantadas. Estas tareas no obedecen la expresión *requirements*.

Originalmente creado para tareas del tipo meta-scheluder como condor_dagman, el Universo Scheluder también puede usarse para manejar tareas de cualquier clase que deban ejecutarse en las máquinas submit.

El Universo Scheluder no usa un demonio condor_starter para manejar la tarea, lo que limita las características y las políticas que soporta. El Universo Scheluder es la mejor opción que deben ejecutarse en la máquina Local, ya que ofrece una amplia variedad de características, y es más consistente que otros Universos, como puede ser el Vanilla. El Universo Scheluder puede retirarse posteriormente, a favor del nuevo Universo local.

2.2.4.7 Universo Paralelo

Permite la ejecución de tareas paralelas, como trabajos MPI, para que sea ejecutados a través del entorno adecuado de Condor.

2.2.4.8 Universo Local

Permite a las tareas de Condor ser lanzadas y ejecutadas con diferentes suposiciones sobre las condiciones de ejecución de la tarea. La tarea no debe esperar a que se le asigne una máquina, ya que se ejecuta en la máquina donde se lanzó la tarea. El trabajo no podrá adelantarse. Los requisitos de las máquinas no se consideran en el Universo Local.

Resumiendo, El Universo Standard permite la migración de las tareas, pero tiene algunas restricciones sobre las tareas que puede ejecutar. El Universo Vanilla permite pocos servicios, pero posee muy pocas restricciones. El Universo PVM es para programas inscritos a la interfaz Parallel Virtual Machine. El Universo MPI es para programas inscritos a la interfaz MPICH. El Universo MPI debe ser supervisado por el Universo Parallel. El Universo Gris permite a los usuarios lanzar tareas usando la interfaz de Condor. El Universo Java permite a los usuarios ejecutar tareas escritas para la máquina virtual de java (JVM). El Universo Scheluder permite lanzar tareas de pequeño tamaño manejadas por el demonio **condor_schedd** en el host donde se haya lanzado. El Universo Parallel es para programas que requieren múltiples máquinas para un mismo trabajo.

Capítulo 3
Instalación y puesta en marcha de Condor Pool

3.1 Pasos previos a la instalación

Antes de realizar la instalación, hay una serie de decisiones que se deben tomar, acerca del manejo de este Pool. Las decisiones responden a estas preguntas:

- **¿Qué máquina será el central manager?**

Una máquina del Pool debe de ser el Central Manager. Hay que instalar Condor en esta máquina primero. Éste será el núcleo de información central del Pool, así como la máquina que negociará entre máquinas que pueden ejecutar las tareas y las tareas que se lazan. Si el Central Manager falla, las tareas del sistema continuarán ejecutándose, pero ninguna otra podrá ser lanzada. Debido a la importancia de esta máquina para el correcto funcionamiento de Condor, el Central Manager debe ser una máquina que o bien nunca se apague, o bien sea capaz de reiniciarse de manera rápida si falla.

También es necesario considerar el tráfico de la red y el retardo de la misma cuando se selecciona el Central Manager. Todos los demonios mandan continuamente mensajes esta máquina. La necesidad de memoria en el Central Manager es mayor cuantas más máquinas haya en el Pool. Del mismo modo, una CPU más rápida mejorará el tiempo de negociación.

- **¿Qué máquinas podrán lanzar tareas?**

Condor puede restringir las máquinas que pueden lanzar tareas. Del mismo modo puede permitir a cualquier máquina de la red conectarse como una máquina Submit para poder lanzar tareas. Si el Pool de Condor se encuentra tras un firewall y todas las máquinas se encuentran detrás de dicho firewall, la configuración de entrada HOSTALLOW_WRITE debe establecerse con el valor *.

Además, es necesario reflejar el conjunto de máquinas a los que se les permite lanzar tareas. Condor trata de ser seguro, por ello, por defecto, no se permite a ninguna máquina lanzar tareas. Para evitar esta situación, basta con modificar en el fichero de configuración la entrada: YOU_MUST_CHANGE_THIS_INVALID_CONDOR_CONFIGURATION_VALUE

- **¿Debe Condor funcionar como Root o no?**

Se deben iniciar los demonios de Condor como usuario Root de Unix. Si no se inicia como Root Condor puede hacer muy poco en cuestiones de seguridad y políticas. Se puede instalar Condor como cualquier usuario, sin embargo esto conlleva serios problemas en seguridad y rendimiento.

- **¿Quién debe administrar Condor?**

Cualquier Root administrará Condor de manera directa, o alguien puede actuar como el administrador de Condor. Si el Root ha delegado la responsabilidad a otra persona, pero no desea darle a esa persona acceso de Root, el Root puede

especificar un fichero *condor_config.root* que permite sobrescribir acciones de los otros ficheros de configuración. De este modo, el fichero de configuración global *condor_config* puede ser controlado por quien sea condor_admin, y el *condor_config.root* sólo es controlado por el Root. Aquellos elementos que pueden comprometer la seguridad del Root (como los binarios lanzados como Root) pueden ser incluidos en el *condor_config.root* mientras que aquellos que sólo que se encargan de las políticas específicas de condor pueden ser controladas sin el acceso de Root.

- **¿Es necesario tener un usuario Condor de Unix? ¿Donde debe localizarse su directorio home?**

Para simplificar la instalación de Condor, crear un usuario Unix llamado Condor en todas las máquinas del Pool. Los demonios de Condor crearán ficheros (como los ficheros log) propios de cada usuario, y el directorio de Condor puede ser usado para especificar la localización de los ficheros y directorios necesarios para Condor. El directorio */home* de un usuario cualquiera puede ser compartido entre todas las máquinas del Pool, o puede ser un directorio */home* separado en la partición local de cada máquina. Ambas opciones tienen ventajas y desventajas. Tener los directorios centralizados hace que la administración sea más sencilla, pero también concentra el uso de recursos que potencialmente necesitan una gran cantidad de espacio, en un solo directorio */home* compartido.

Si no se crea un usuario llamado Condor de manera específica, entonces debe especificarse cualquiera que sea a través de la entrada de configuración CONDOR_IDS.

- **¿Dónde deben situarse los directorios específicos de Condor en la máquina?**

Condor necesita unos pocos directorios que son únicos en cada máquina del Pool. Dichos directorios son */sPool, /log* y */execute.* Generalmente estos tres directorios, son los subdirectorios del directorio local de la máquina (especificadas en la macro LOCAL_DIR en el fichero de configuración). Cada directorio local pertenece al usuario donde actúa Condor.

Si se posee un usuario Unix llamado Condor con un directorio */home* local en cada máquina, el LOCAL_DIR podría ser el directorio */home* de Condor (LOCAL_DIR=$(TILDE) en el fichero de configuración). Si lo que se tiene es un directorio */home* compartido entre todas las máquinas del Pool, se podría querer crear un directorio para cada host (indicado por su hostname) para el directorio local (por ejemplo, LOCAL_DIR=$(TILDE)/hosts/($HOSTNAME)). Si no se posee una cuenta específica de Condor en las máquinas, se pueden poner los directorios donde se desee:

Execute

Este es el directorio que actúa como el directorio de trabajo para cualquier tarea de Condor que se ejecute en una máquina Execute. El binario del trabajo remoto es copiado dentro de este directorio, por lo que debe haber suficiente espacio para ello (Condor no debe enviar una tarea a una máquina que no posea suficiente espacio en memoria para contener el binario inicial). De la misma manera, si el trabajo remoto es eliminado por cualquier razón, primero se elimina del directorio */execute*. Por ello es necesario poner el directorio */execute* en una partición con suficiente espacio de memoria.

Spool

Almacena la cola de trabajo y los ficheros históricos, así como los ficheros checkpoint de todas las tareas sometidas en dicha máquina. Como resultado los necesidades de espacio de disco pueden ser grandes,

especialmente si los usuarios lanzan tareas con ejecutables o tamaño de imagen muy grande. Usando un servidor Checkpoint, uno puede mejorar las necesidades de espacio, p.ej. todos los ficheros del Pool se almacenan en el servidor, en lugar de los directorios */sPool* de cada máquina. Sin embargo los ficheros checkpoint iniciales se almacenan en el directorio */sPool*, por lo que se necesita algo de espacio.

Log

Cada demonio de Condor escribe su propio fichero log, y cada fichero log está localizado en el directorio */log*. Se puede especificar que tamaño se desea para esos ficheros, ya que los requisitos del directorio son configurables. Si se posee un sistema de ficheros distribuido instalado en el Pool, se puede querer situar los directorios */log* en una dirección común (como /usr/local/condor/logs/$(HOSTNAME)), por lo que se pueden ver los ficheros log de todas las máquinas en una única dirección. Sin embargo, para ello se debe especificar una partición local para el directorio */lock*.

Lock

Condor usa un pequeño número de ficheros lock para sincronizar el acceso a determinados ficheros que son compartidos entre múltiples demonios. Debido a los problemas encontrados con el fichero locking y los sistemas de ficheros distribuidos, los ficheros lock deben de situarse en una partición local de cada máquina. Por defecto se encuentra en el directorio */log*. Si se sitúa el directorio */log* en una partición de un sistema de ficheros, especificar una partición para los ficheros lock con el parámetro LOCK del archivo de configuración.

Generalmente, se recomienda no establecer los directorios antes comentados (excepto el directorio */lock*) en la misma partición, como /var. Esto puede causar muchos problemas para las máquinas. Idealmente se debe tener una partición separada para los directorios Condor. Por tanto, la única consecuencia de no situar bien los directorios, es el mal funcionamiento de Condor, no de la máquina entera.

- **¿Dónde deben de situarse los diferentes componentes de Condor?**

-Ficheros de configuración

Hay una serie de ficheros de configuración que permiten diferentes niveles de control sobre como Condor está configurado en cada máquina del Pool. El fichero de configuración global está compartido por todas las máquinas del Pool. Por razones administrativas, este fichero debería estar localizado en un sistema de ficheros compartido si es posible. Además hay un fichero de configuración global para cada máquina, donde se pueden sobrescribir características del fichero global. Esto permite tener diferentes demonios activos, diferentes políticas sobre cuando empezar y parar Condor, y más aún. También se puede tener ficheros de configuración específicos para cada plataforma del Pool. También se recomienda empezar los demonios como Root; esto permite crear ficheros de configuración manejados por el Root, que sobrescribirán a los demás. De este modo, si el administrador de Condor no funciona como Root, los ficheros de configuración de Condor puede pertenecer y ser modificado por el administrador de Condor, pero el Root no tiene por qué garantizar al acceso de Root a esa persona.

En general, hay un número de lugares en el que Condor buscará los ficheros de configuración. El primero por el que buscará, será el fichero de configuración global. Las posibles localizaciones son descartadas en orden hasta que el fichero de configuración es encontrado. Si no contiene un archivo de configuración válido, Condor mostrará un mensaje de error y finalizará:

1. El fichero estará especificado en la variable de entorno CONDOR_CONFIG.
2. /etc/condor/condor_config
3. condor/condor_config
4. $(GLOBUS_LOCATION)/etc/condor_config

Si uno especifica un fichero en la variable de entorno CONDOR_CONFIG y hay un problema leyendo este fichero, Condor mostrará un mensaje de error y saldrá, sin continuar buscando en el resto de opciones. Sin embargo si no se especifica un fichero en la variable CONDOR_CONFIG, Condor buscará en las otras opciones.

A continuación Condor trata de cargar el fichero de configuración local. La única manera de especificar el fichero de configuración local, es en el fichero de configuración global, en la macro LOCAL_CONFIG_FILE. Si esa macro no está definida no se usa ningún fichero de configuración local. Esta macro puede se un único fichero o una lista.

La última en ser buscada es el archivo de configuración de Root. El fichero global es buscado en los siguientes sitios:

1 /etc/condor/condor_config.root

2 condor/condor_config.root

El fichero de configuración local de Root se encuentra en la macro LOCAL_ROOT_CONFIG_FILE. Si no se especifica esta variable este fichero no es usado. Esta variable puede ser un único fichero o una lista de los mismos

-Directorio Release

Cualquier distribución binaria contiene un fichero release.tar que contiene 5 subdirectorios: bin, etc, lib, sbin y libexec. Donde decida instalarse estos 5 directorios será el directorio realease (especificado en la macro RELEASE_DIR del fichero de configuración). Cada directorio release contiene unos binarios y librerías dependientes de la plataforma, por eso necesitas instalar una según cada máquina del Pool. Para facilitar la administración, esos directorios deben localizados en un sistema de archivos compartidos, si es posible.

-Binarios de usuario:

Todos los ficheros del directorio *bin* son programas que los usuarios de condor deben de tener en su ruta. Se pueden poner en una localización conocida (como /usr/local/condor/bin) la cual debe ser añadida por los usuarios Condor a su variable de entorno PATH, o copiar estos ficheros directamente dentro de un lugar conocido en el PATH de usuario (como usr/local/bin). También se pueden situar los binarios en /usr/local/bin haciendo enlaces simbólicos a /usr/local/bin.

-Binarios de sistema:

Todos los ficheros del directorio *sbin* son demonios de Condor y agentes, o programas que sólo el administrador de Condor necesita que funcionen. Hay que añadir esos programas sólo al PATH del administrador de Condor.

-Binarios privados de Condor:

Todos los ficheros del directorio *libexec* son programas Condor, que nunca deben se accionados manualmente, pero que son usados de manera interna por Condor.

-Directorio lib:

Los ficheros en el directorio lib son las librerías de Condor que deban ser lincadas, que serán usadas por las tareas migratorias y de checkpointing de los usuarios Condor. Lib también contiene escritos usados por **condor_compile** para ayudar a lincar los trabajos con las librerías de Condor. Estos ficheros deben estar en una localización que sea accesible para todo el mundo, pero no tiene por qué estar en el PATH de todos los usuarios. **Condor_compile** busca en el fichero de configuración para localizar el directorio lib.

-Directorio etc

Contiene un subdirectorio de ejemplos el cual contiene ejemplos de ficheros de configuración y otros ficheros usados para instalar Condor. Etc es la localización recomendada para guardar una copia master de los ficheros de configuración. Se pueden crear enlaces en cualquiera de las rutas anteriormente mencionadas sobre las que condor buscará automáticamente para buscar el fichero de configuración global.

-Documentación

La documentación dada por Condor, se puede encontrar en HTML, postscript y PDF. Se puede encontrar la documentación de condor en: http://www.cs.wisc.edu/condor/manual

- **¿Estoy usando AFS?**

Condor no posee una manera directa para autenticarse por si mismo a AFS. Esto supone que no se va a poder tener el LOCAL_DIR de condor en AFS. Sin embargo se puede tener el directorio RELEASE_DIR en AFS, con lo que se puede tener una copia de esos ficheros, y actualizarlos a través de una localización central.

- **¿Tengo suficiente espacio libre para Condor?**

Condor ocupa un espacio de disco considerable. Esta es otra razón por lo que es una buena idea tenerla en un sistema de archivos compartido. Las necesidades de espacio para las descargas, son dadas en la página de descargas. Además se debe tener suficiente espacio de disco en el directorio local de las máquinas que ejecutan tareas para Condor.

3.2 Condor Pool en S.O Windows.

3.2.1 Entorno del Condor Pool

Una vez explicadas las características más importantes de Condor, se procede a la instalación del mismo en una red prototipo.

Aunque la red que se muestra a continuación está compuesta por cuatro máquinas, la idea es que un futuro se proceda a su instalación en los laboratorios del área de ingeniería de telemática, para con ello aprovechar los recursos que hoy en día se están desaprovechando, como ya se explicó en la introducción de este proyecto.

Esta red prototipo está compuesta por cuatro máquinas y un hub, cuyas características se describen a continuación:

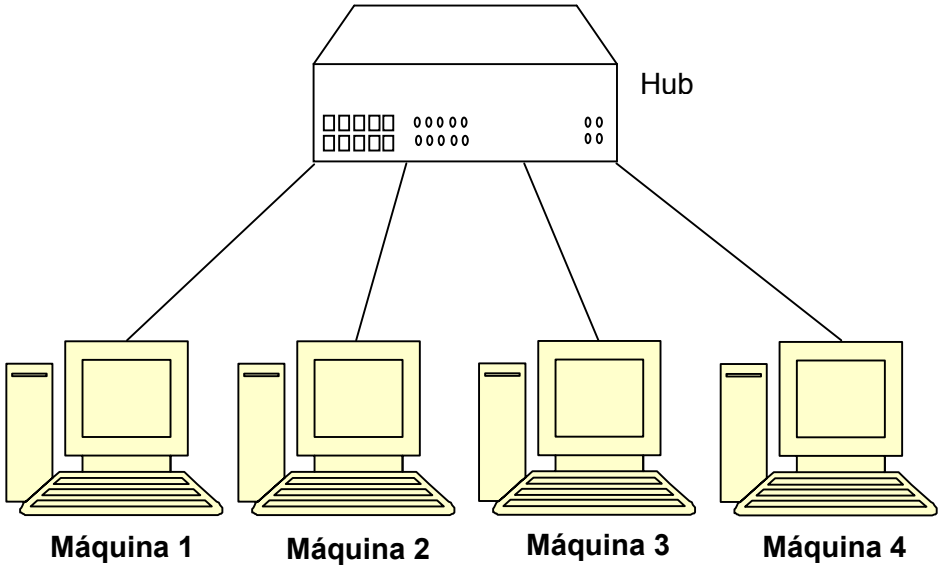


Figura 2. 3 Esquema de la red prototipo

Máquina	Dirección	Papel
1	192.168.1.1	Central Manager: execute (cada 15 minutos) y submit
2	192.168.1.2	execute (siempre) y submit
3	192.168.1.3	submit
4	192.168.1.4	execute y submit

Tabla 3. 1 Papeles que desempeñan las máquinas de la red prototipo

Se ha optado por la instalación del Universo Vanilla en todas las máquinas. El principal motivo de esta elección, fue el hecho de que las tareas del Universo Standard, al generar ficheros Checkpoint, necesitan una máquina (servidor Checkpoint), que se encargue del almacenamiento de esto ficheros. Esto supone un gran gasto de memoria, lo que a la larga hace que se desperdicien recursos de las máquinas. El Universo Vanilla, pese a no realizar checkpointing, es el más consistente de los posibles Universos. Ofrece fiabilidad y robustez. Las tareas, si se suspenden, permanecerán almacenadas en la máquina, y continuarán su ejecución cuando la máquina quede libre, sin perderse tiempo de ejecución. Además este universo es el más sencillo de adaptar a las posibles, que se deseen llevar a cabo en el Pool.

3.2.2 Requisitos Hardware

Las características hardware empleado en la realización del proyecto son las siguientes:

Función	Marca	Arquitectura	Memoria	Disco Duro
Central Manager,Submit y Execute	HP Pavilion	Intel x86	512 MB	80 GB
Submit	HP Pavilion	Intel x86	512 MB	80 GB
Execute	HP Pavilion	Intel x86	512 MB	80 GB
Execute	HP Pavilion	Intel x86	512 MB	80 GB

Tabla 3. 2 Características principales de las máquinas de la red prototipo

La arquitectura hardware de la red se basa en un Hub 3Com de 8 puertos, los ordenadores conectados a él a través de cables Ethernet directos categoría 5 UTP, toda la red a velocidad de 100 MB/s.

Cabe destacar que las características de este hardware en particular han limitado en gran medida, el software y las tareas que se pueden ejecutar.

3.2.3 Instalación del software Condor

Antes de la instalación es aconsejable crearse una nueva cuenta de Windows (en este caso condor con el password condor), para de esa manera tener almacenadas las tareas en un lugar distinto a nuestra cuenta habitual de Windows, y sólo iniciar el servicio cuando realmente se necesite. Si no se hace, será necesario asegurarse de que la cuenta posee una clave de acceso (y si no la posee crearla), pues en pasos posteriores se pedirá.

La instalación de Condor debe hacerse con privilegios de administrador. Después de la instalación, los servicios de Condor funcionarán en el sistema local. Cuando Condor ejecuta un trabajo lo ejecutará con los permisos de usuario.

El primer paso para realizar la instalación es descargar Condor de la página oficial: http://www.cs.wisc.edu/condor/downloads. Para poder proceder a la descarga, primero hay que registrarse. Una vez hecho esto, Condor remite a la página de descargas, donde se seleccionarán los binarios más adecuados para el hardware que se posea (indicado anteriormente). Cuando de haya descargado podrá comenzar el proceso de instalación haciendo doble clic en el icono que aparece tras la descarga. La instalación de Condor se completa resolviendo las preguntas, y escogiendo las opciones de los siguientes pasos:

Condor ya está instalado

Si Condor ya ha sido instalado en el ordenador, un cuadro de diálogo aparecerá antes del proceso de instalación de Condor. Se preguntará si se desean guardar, los ficheros actuales de configuración de Condor.

Figura 3. 1 Opciones de la instalación en Windows

Si se responde que si, los ficheros de configuración no se cambiarán, y el proceso continuará en el punto donde los nuevos binarios serán instalados. Si la respuesta es no, entonces se preguntará si se desean usar las respuestas dadas en la anterior instalación.

Paso 1: Acuerdo de licencia

El primer paso de la instalación es la bienvenida y el acuerdo de licencia. Se debe tener en cuenta que es mejor llevar a cabo la instalación, sin que ningún otro programa de Windows esté funcionando. Si se necesita cerrar algún programa, es más seguro cancelar el programa de instalación y cerrar el programa. Se debe aceptar la licencia. Se puede responder si o no. Si no se está de acuerdo con la licencia la instalación no continuará. Una vez que se acepte la licencia, la siguiente ventana de la instalación es para que se ponga el nombre e información de la compañía, o usar los que se dan por defecto.

Paso 2: Configuración del Condor Pool

La instalación de Condor requiere información, que depende de si el usuario está creando un nuevo Pool, o se une a un Pool existente. Si se está creando un nuevo Pool, la instalación requiere que la máquina donde se realice la instalación sea el Central Manager. Para la creación de un nuevo Pool, se preguntará sobre información acerca del nuevo Pool:

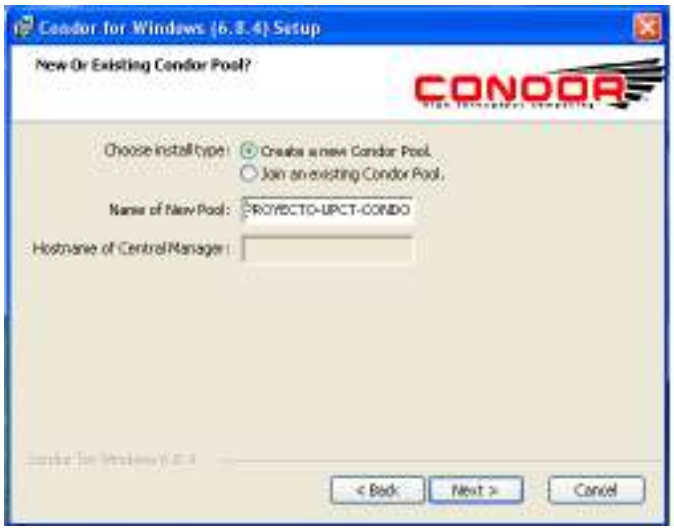


Figura 3. 2 Creación de un nuevo Pool en Windows

Si existe un Pool existente, el programa de instalación requiere el hostname del Central Manager. En este caso el central Manager se encuentra en la dirección ip 192.168.1.1:

Figura 3. 3 Unirse a un Pool existente en Windows

Paso 3: El papel de la máquina

Este paso es suprimido en la instalación de un Condor personal. Cada máquina de un Condor Pool puede lanzar tareas, o bien ejecutarlas. Este paso de la instalación permite elegir si la máquina sólo va a lanzar tareas (submit), sólo va a ejecutar tareas (execute), o ambas. El caso más común es ambos (el valor por defecto).En este caso los papeles de las diferentes máquinas son los siguientes:

Máquina	Dirección	Papel
1	192.168.1.1	Central Manager: execute (cada 15 minutos) y submit
2	192.168.1.2	execute (siempre) y submit

3	192.168.1.3	Submit
4	192.168.1.4	execute y submit

Tabla 3. 3 Asignación de direcciones a las máquinas de la red prototipo

Paso 4: ¿Donde deberá instalarse Condor?

Este paso decide donde se situarán los ficheros de Condor. En este caso en particular se seleccionó la instalación por defecto en c:\condor. La instalación en el disco local se decide por varias razones:

- Los servicios de Condor funcionan en el sistema local, y a través de Microsoft Windows, el sistema local no posee privilegios de red. Por lo tanto, para que funcione Condor, Condor debe instalarse en una partición de disco duro, a contrario de una partición de red (servidor de ficheros).
- La segunda razón para la instalación en un disco local es que el uso de Windows, posee implicaciones, dependiendo de donde se instale Condor.
- Es posible situar Condor en un disco local que no sea local, si esta dependencia es añadida a los servicios del control manager, por lo que Condor comienza después de que los servicios de ficheros requeridos sean correctos.

Paso 5: ¿Dónde se encuentra la máquina virtual de Java?

Es posible para Condor ejecutar trabajos en el Universo Java. Para poder tener soporte para java se debe facilitar un path para java.exe al sistema. Si se quiere deshabilitar el Universo de java, solo hay que dejar esa línea en blanco. En este Pool en particular se dejó el valor por defecto, para posibles posteriores usos del Universo de Java.

Figura 3. 4 Universo Java en Windows

Paso 6: ¿Dónde debe Condor mandar un e-mail si las cosas van mal?

Varias partes de Condor mandan correos al administrador de Condor, si algo va mal, y requiere atención humana. La dirección de e-mail y el SMTP de ese administrador.

Figura 3. 5 Servidor SMTP y dirección del administrador de Condor

Paso 7: El dominio

Este paso es eliminado en la instalación de un Condor personal. En el Pool de este proyecto se usará el dominio de la universidad.

Figura 3. 6 Dominio DNS o UID de la máquina

Paso 8: Permisos de acceso

Este paso se omite en la instalación de un Condor personal. Las máquinas del Pool de Condor, necesitarán varios tipos de permisos de acceso. Las tres categorías son:

Lectura: permite a una máquina obtener información acerca de Condor, así como del estado de las máquinas de Pool y las colas de trabajo. Todas las máquinas del Pool deben de tener acceso de lectura. Además, dar acceso de lectura a *.cs.wisc.edu permitirá al equipo de Condor obtener información sobre el Pool de Condor, si dicha información es necesaria.

Escritura: Todas las máquinas del Pool pueden tener permisos de escritura. Esto permite a las máquinas que se especifique mandar información a los demonios de Condor, por ejemplo, empezar un trabajo de Condor. Para que una máquina se una a un Pool de Condor, debe tener permisos de lectura y escritura por parte de todas las máquinas del Pool.

Administrador: Una máquina con acceso de administrador, poseerá permisos extendidos, para poder realizar cosas como: cambiar las prioridades de los usuarios, modificar la cola de trabajo, iniciar y parar los servicios, y reiniciar Condor. Se le deberá dar acceso de administrador al Central Manager, que es la opción por defecto. Este acceso estará garantizado para toda la maquina, por lo que habrá que tener cuidado.

Figura 3. 7 Permisos de administrador, lectura y escritura en Windows

En este caso, todas las máquinas del Pool, tendrán acceso de lectura y escritura, y cada máquina poseerá para sí misma permisos de Root.

Paso 9: Políticas de Trabajo

Condor ejecutará las tareas sometidas en las máquinas, en un orden de preferencia dado por la instalación. Se dan tres opciones, siendo la primera la más comúnmente usada por Condor. Esta especificación puede cambiarse, según las necesidades de las máquinas.Las tres *opciones* son:

- Después de 15 minutos de no haber actividad en la consola y baja actividad de la CPU.
- Ejecutar siempre tareas Condor.
- Después de 15 minutos de no haber actividad en la consola.

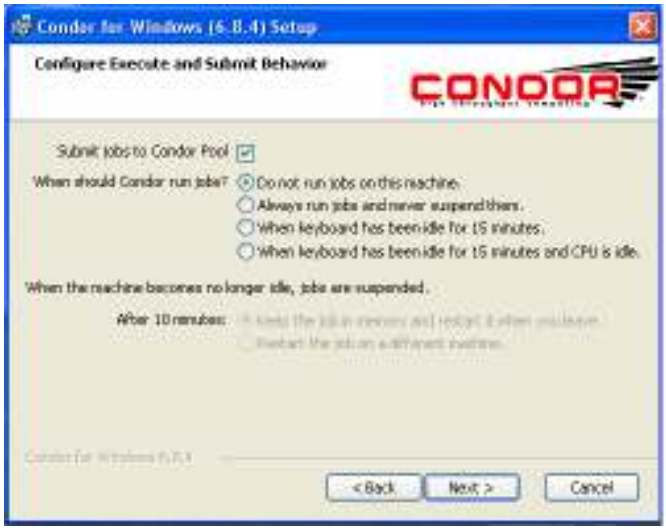


Figura 3. 8 Políticas de trabajo en Condor

Con actividad de la consola se refiere al uso del ratón o del teclado. Por ejemplo si uno está leyendo este documento online, y usa para ello tanto el ratón como el teclado para cambiar la posición del texto, se está generando actividad de consola.

Actividad baja de CPU se considera un porcentaje de carga inferior al 30% (lo cual puede ser modificado en el archivo *condor_config*). Si se posee una máquina con múltiples procesadores, este porcentaje se considera como la media de actividad de CPU de todos los procesadores. Si lo que se quiere es hacer una prueba general del funcionamiento de Condor, la mejor opción es la de ejecutar tareas siempre. La elección que se hizo para las máquinas de este caso fue:

Máquina	Papel	Política de Trabajo
1ª	Central Manager: Submit y Execute	Ejecutar tareas después de 15 minutos sin actividad de consola
2ª	Execute y Submit	Ejecutar tareas siempre
3ª	Submit	No ejecuta tareas de Condor (solo las lanza)
4ª	Execute y Submit	Ejecutar tareas siempre

Tabla 3. 4 Políticas de trabajo de las máquinas de la red prototipo

Paso 10: Política de eliminación de tareas

Este se paso se omite si se elige la opción (Paso 9) de que las tareas de Condor se ejecuten siempre en las máquinas. Si Condor está ejecutando una tarea, y el usuario vuelve a usar la máquina, Condor inmediatamente suspenderá el trabajo, y después de 5 minutos decidirá que hacer con el trabajo que se ha completado parcialmente. Hay dos opciones:

- *La tarea se elimina 5 minutos después de que el usuario regrese:* El trabajo se suspende inmediatamente cuando se detecta actividad de consola. Si la actividad de consola continúa durante 5 minutos, entonces el trabajo se elimina. Como en la versión para Windows no se incluye la

opción de check-pointing, la tarea deberá comenzar su ejecución desde el principio cuando se reinicie la ejecución. La tarea será puesta de nuevo en la cola.

- *Suspender el trabajo dejándolo en memoria:* La tarea se suspende inmediatamente. Después, cuando la actividad de la consola cese durante 10 minutos, la ejecución de la tarea de Condor será reanudada. El inconveniente de esta opción, es que en el momento que el trabajo debe permanecer en memoria, ocupará espacio de la misma. En determinadas ocasiones, sin embargo, este espacio de memoria en muy pequeño.

¿Qué opción elegir? Matar una tarea es menos intrusito, que dejarlo en memoria durante algún tiempo. Dejarlo en memoria, sin embargo, tiene el beneficio de que no se pierde el tiempo de ejecución que ya se ha utilizado para dicha tarea.

En el caso del Pool de este proyecto, como el objetivo es que muchas máquinas puedan ejecutar el servicio ofrecido por Condor, se ha optado por la primera opción, con el objetivo de no colapsar la memoria de las máquinas.

Paso 11: Revisión de la información introducida

Por último se recomienda hacer un repaso de los pasos anteriores, para asegurarse de que las opciones que se han seleccionado son las que interesan. Siempre se tiene la opción de novel hacia atrás en el cuadro de diálogo. Una vez finalizados estos pasos, antes de finalizar la instalación, se preguntará si se desea iniciar el servicio de Condor, a lo que se contestará que si. El directorio *c:\condor* ya ha sido creado así como los subdirectorios y archivos correspondientes.

3.2.4 Puesta en marcha de los demonios Condor.

Una vez finalizada la instalación de Condor, los demonios se inicializan de manera automática, por el propio programa de instalación. Si se quiere visualizar los demonios, puede hacerse mediante *ctrl.+alt+supr*, para visualizar en el administrador de sistema los demonios de Condor activos. En el caso de la *Figura 3.2.4.13* se puede observar los demonios master, schedd y startd por tratarse de una máquina Execute y Submit:

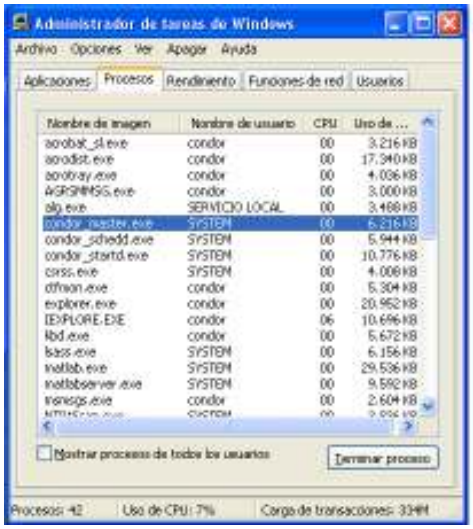


Figura 3. 9 Comprobación del funcionamiento de los demonios de Condor en Windows

Si la máquina en cuestión fuera el Central Manager deberían verse también los demonios negotiator y collector**.** Si por cualquier motivo no se hubieran iniciado los demonios no hubieran arrancado de manera automática, deberán arrancarse de manera manual. Hay tres maneras de arrancar el servicio de Condor

1. Reiniciando la máquina.
2. Panel de Control→Herramientas_Administrativas→Servicios→Condor→Start
3. En la línea de comandos, introduciendo *net start condor*

Una vez hecho esto el servicio de condor se iniciará de manera automática cada vez que se inicie la máquina.

Cuando el servicio de Condor se haya iniciado, lo siguiente que deberá hacerse es añadir nuestras credenciales, para poder empezar a lanzar tareas al Pool. Para hacerlo debe usarse el comando de Condor *condor_store_credd add*. Una vez introducido dicho comando se pedirá por pantalla que se introduzca un password. Ahí debe introducirse la clave de la propia cuenta de Windows, que en este caso como se indicó al inicio es condor. Si el password es correcto, Condor indicará que la operación se ha completado correctamente.

Inicialmente sólo pueden usarse los comandos de Condor desde la ruta *c:\condor\bin.* Para cambiar esto será necesario modificar la variable de entorno *PATH,* la cual que encuentra en Mi PC→Propiedades del Sistema→Opciones Avanzadas→Variables de entorno→PATH→Modificar. Sólo se necesita añadir a las rutas que hay, la ruta *c:\condor\bin* y ya se podrán usar los comandos de Condor desde cualquier lugar.

Figura 3. 10 Modificación de la variable PATH

3.3 Condor Pool en S.O Linux.

3.3.1 Entorno del Condor Pool

El entorno del Pool de Condor es el mismo que el que se indicó en el apartado 3.2.1 de la instalación del Condor en Windows.

3.3.2 Requisitos Hardware

Las características del hardware, son exactamente las mismas que las que se describieron en el apartado 3.2.2.

3.3.3 Requisitos Software

Antes de comenzar la instalación del Pool, es necesario llevar a cabo como Root, la instalación de NIS y NFS en las máquinas que forman parte del Pool.

3.3.3.1 Instalación de NIS

Para ejecutar el software de servidor/cliente de NIS se necesitará ejecutar antes el programa *portmap*. En RedHat Linux existe un script en */etc/init.d/portmap* para arrancar éste demonio. Todo lo que tienes que hacer es añadirlo a un runlevel si deseas que se realice en el arranque.

El mapeador RPC es un servidor que convierte números de programas RPC en números de puerto de protocolo TCP/IP (o UDP/IP). Debe estar ejecutándose para poder realizar llamadas RPC (que es lo que el software de cliente NIS hace) a servidores RPC (como un servidor NIS) de esa máquina. Cuando un servidor RPC arranca, avisará al mapeador de puertos por cuál puerto está escuchando, y a qué números de programas RPC está preparado para servir. Cuando un cliente desea hacer una llamada RPC a un número de programa dado, primero deberá contactar con el mapeador de puertos de la máquina servidora para determinar el número de puerto al que los paquetes RPC deben ser enviados. El comando *rpcinfo* nos dará la información de que programas está preparado para servir.

El software necesario para poner en marcha y configurar un servidor o cliente de NIS en una distribución RedHat Linux, es el siguiente:

ypserv

yp-tools

ypbind

Instalación del Cliente NIS

Hay que asegurarse de que están instalados los paquetes ypbind e yp-tools. El programa /usr/sbin/ypbind es el servicio cliente de NIS, que nos permitirá interactuar con un Servidor de NIS y acceder a los mapas que este sirva (/etc/password, /etc/group). Para comprobar que ypbind funciona correctamente, habría que seguir los siguientes pasos:

1. Tener arrancado el portmap para lo que se usará el comando rpcinfo –p

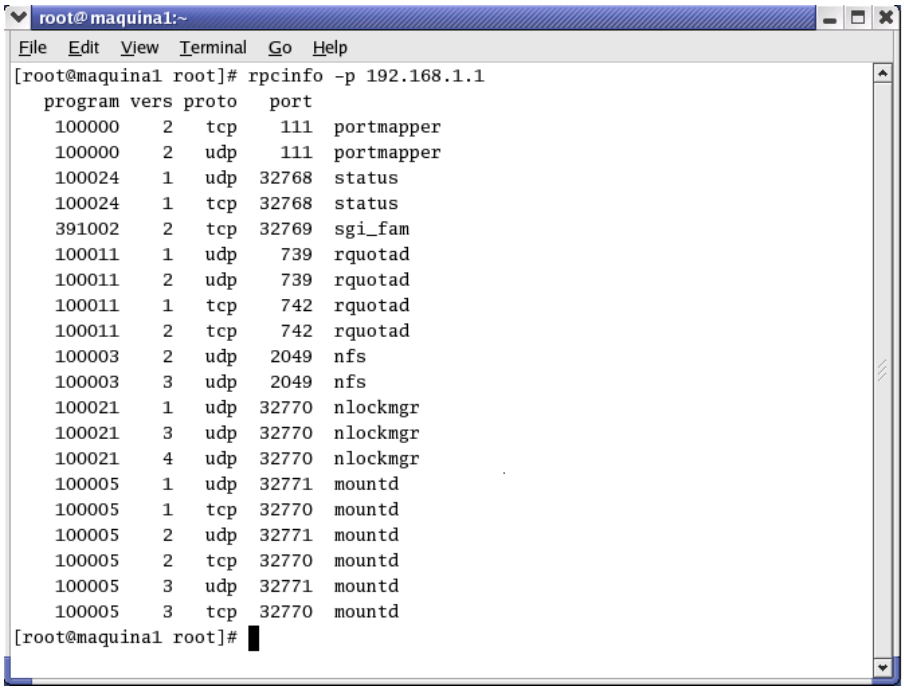


Figura 3. 11 Verificación del funcionamiento del portmap

2. Definir el dominio de NIS al cual sirve el servidor con el comando *domainname*. En nuestro caso el dominio es *upct.es*:

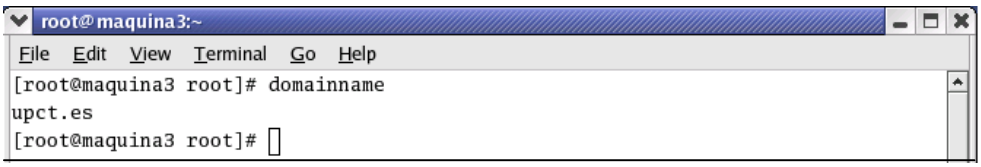


Figura 3. 12 Comprobación del dominio NIS de la máquina

Figura 3.3.3.1. 1

3. Crear el directorio /var/yp si no existe.
4. Iniciar el demonio ypbind /usr/sbinypbind.
5. Usar el comando rpcinfo –p localhost para comprobar si ypbindes capaz de registrar su servicio con el maleador de puertos.

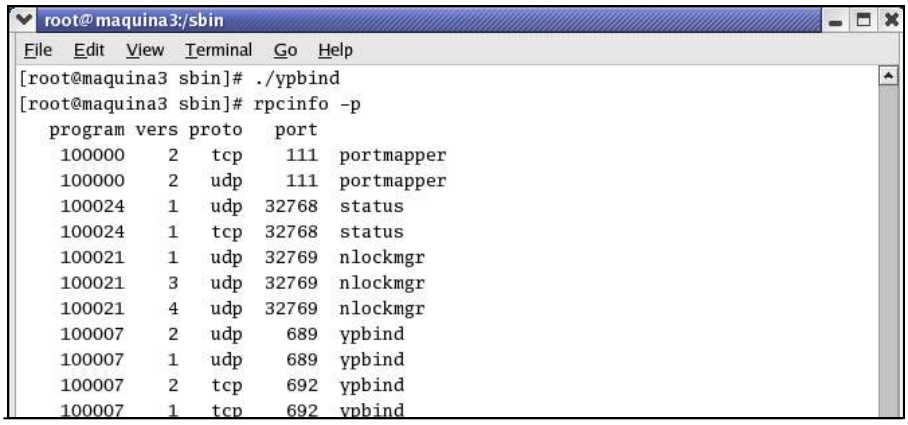


Figura 3. 13 Comprobación del funcionamiento del ypbind

6. Comprobar que puedes acceder a los mapas que sirve el servidor NIS con el comando ypcat passwd. Esto debería devolver el mapa passwd del servidor.

Si todo ha funcionado correctamente, debe automatizarse este proceso para que se realice cada vez que arranque el cliente. Para ello hay que modificar el script */etc/rc.d/init.d/ypbind* que acepta {start|stop} para lanzar o parar el servicio.

Instalación del Servidor NIS

Habrá que comprobar que está instalado el paquete ypserv (*rpm -q ypserv*), que es el que nos permitirá instalar un servidor de páginas amarillas o NIS. Habrá que editar el fichero */etc/ypserv.conf* que nos permitirá configurar algunas opciones para el sevidor ypserv. Normalmente contiene una lista de reglas para hosts especiales y reglas de acceso a los mapas. El fichero */var/yp/securenets* define los derechos de acceso al servidor NIS por parte de los clientes. El fichero contiene pares netmask/network de forma que si la dirección IP del cliente se corresponde con la entrada podrá actuar como cliente de NIS.

Hay que asegurarse que el portmap está corriendo (*rpcinfo -p localhost*) y entonces y solo entonces lanzar el demonio ypserv. El siguiente paso sería generar las bases de datos (mapas) de NIS. En el servidor maestro habrá que ejecutar */usr/lib/yp/ypinit –m*

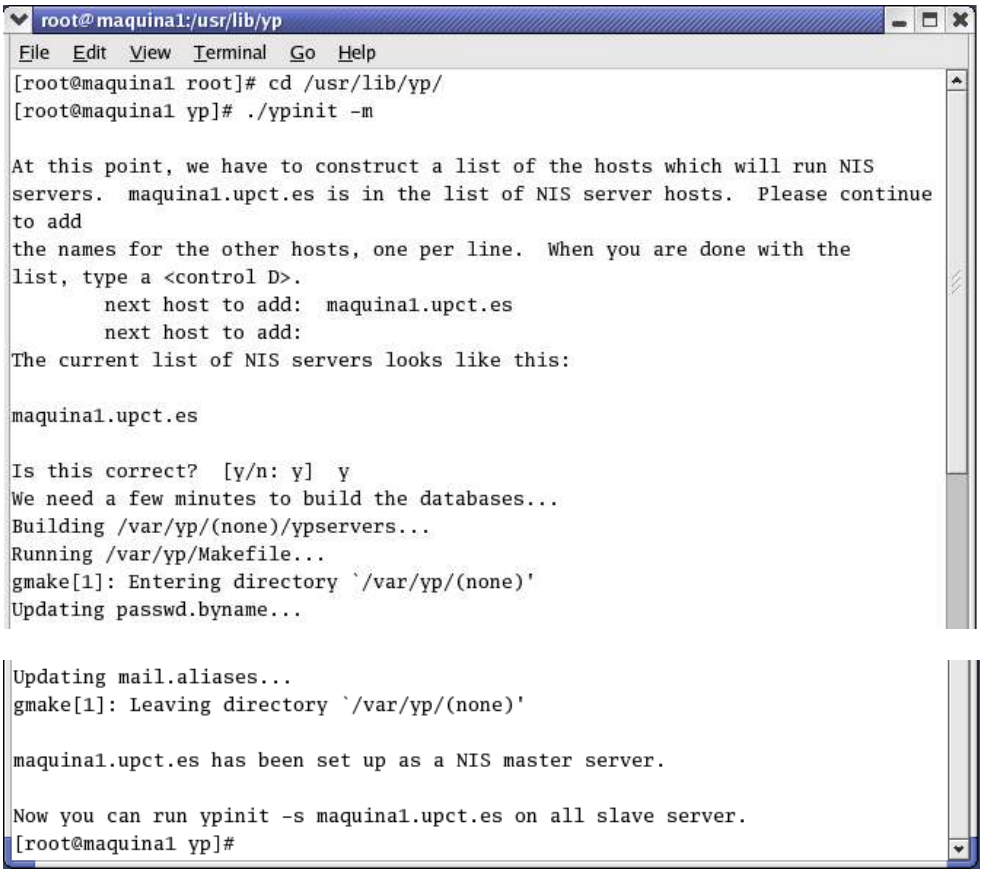


Figura 3. 14 Creación de los mapas de NIS

Por último hay que ejecutar el demonio ypserv , cuyo funcionamiento podrá comprobarse ejecutando rpcinfo –p localhost, debiendo aparecer algo parecido a:

100004 2	udp	965	ypserv
100004 1	udp	965	ypserv
100004 2	tcp	968	ypserv
100004 1	tcp	968	ypserv

3.3.3.2 Instalación de NFS

El *Network File System* (Sistema de archivos de red), o NFS, es un protocolo de nivel de aplicación, según el Modelo OSI. Es utilizado para sistemas de archivos distribuido en un entorno de red de computadoras de área local. Posibilita que distintos sistemas conectados a una misma red accedan a ficheros remotos como si se trataran de locales. Para la instalación necesitaremos tener instalado portmap y el paquete nfs (nfs-utils) que se puede encontrar en la mayoría de las distribuciones en el ordenador que vaya a hacer de servidor de disco. El portmap nos permitirá realizar conexiones RPC al servidor y es el encargado de permitir o no el acceso al servidor a los equipos que especifiquemos. Para saber si se tiene el portmap instalado bastará con un simple

>> *ps aux | grep portmap*

Para saber si NFS está en marcha haremos una consulta al portmap para que nos indique qué servicios tiene en marcha

```
>> rpcinfo -p
```

Cuya salida debería ser algo similar a:

```
100003  2       udp     2049    nfs
100003  3       udp     2049    nfs
```

Para realizar la instalación en el servidor debe hacerse lo siguiente:

1) Botón de inicio → Server Settings → NFS Server
2) Add
3) Se rellena el formulario para agregar el directorio */home* que contendrá a los usuarios, con permisos de *lectura y escritura* y para cualquier ordenador de la red.

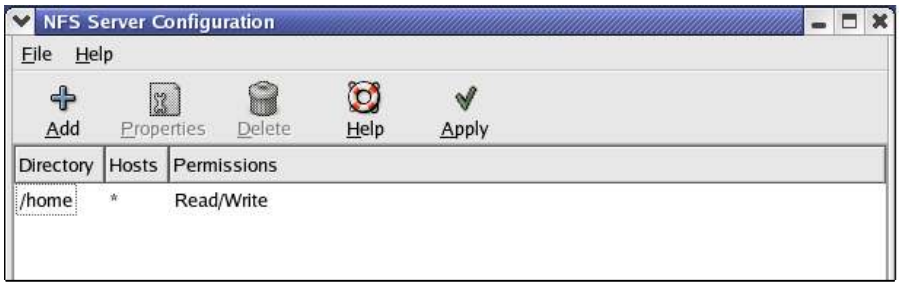


Figura 3. 15 Creación del demonio NFS

En los clientes será necesario editar el directorio /etc/exports, para indicar que se desea compatir, quien es el servidor, y con que permisos se va a compartir:

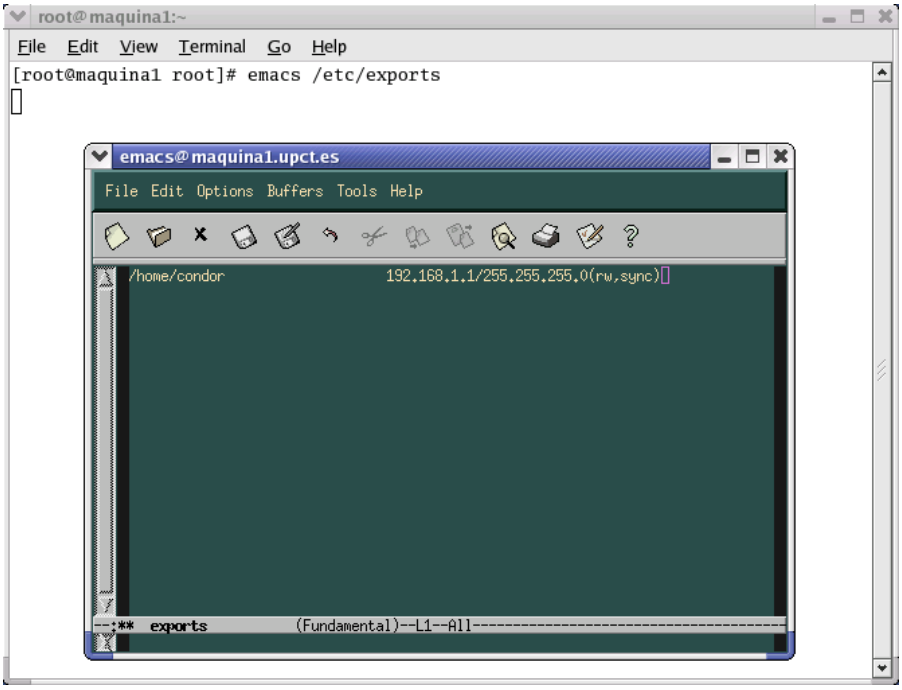


Figura 3. 16 Modificación del fichero exports

En el caso de este proyecto en particular, se decidió compartir el directorio /home/condor, tomando como servidor NFS la Máquina 1, dando permisos de lectura y escritura a los clientes. Otra manera de comprobar el correcto funcionamiento de NFS sería usando la combinación de menús:

1) Botón de inicio → Server Settings → Services
2) Se inicia si fuera necesario.

Figura 3. 17 Activación del demonio NFS

3.3.4 Instalación del software Condor

Una vez se ha instalado NIS y NFS, el primer paso en la instalación será el crearse una cuenta Condor. Para crear una cuenta de usuario nueva en Redhat, ejecutar Menú de Inicio→System Settings→Users and Groups→Add User. En este caso se ha optado por una cuenta con el nombre de condor, la clave condor, y directorio, local en */home/condor*. De este modo ya se está preparado para proceder a la instalación de condor.

El primer paso para realizar la instalación es descargar Condor de la página oficial: http://www.cs.wisc.edu/condor/downloads. Para poder proceder a la descarga, primero hay que registrarse como ya se explicó en el punto 3.2.4. En este caso se bajó *condor-6.8.5-linux-x86-redhat80.tar.gz* que era el más adecuado a la arquitectura de las máquinas como ya se indicó anteriormente.

Después de la descarga, primero deben descomprimirse los archivos. Para ello es necesario ejecutar en el directorio */root*:

tar -xzf linux-x86-redhat80.*tar.gz*

Una vez descomprimido podrán observarse los siguientes archivos:

- DOC Direcciones donde se puede encontrar información sobre Condor.
- INSTALL Direcciones de instalación.
- LICENSE.TXT La licencia de Condor
- README Información general de Condor.
- *condor_install* El script Perl usado para instalar y configurar Condor.
- examples/ Directorio que contiene ejemplos de programas en C,C++ y Fortran, para ejecutar en Condor.
- *release.tar* fichero que contiene los binarios y las librerías de condor

Gracias a esto se crearán los subdirectorios */bin, /etc, /include, /sbin, /lib,* como se observa en la siguiente figura:

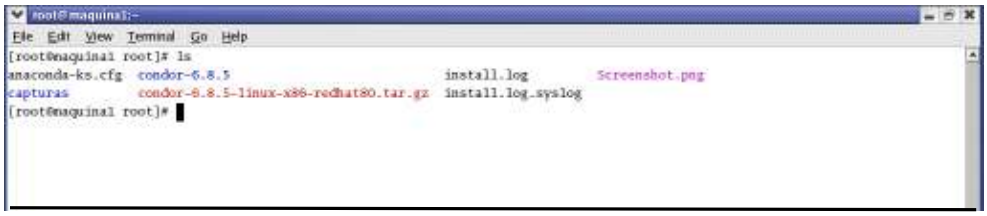


Figura 3. 18 Descomprimir el archivo de instalación

Hay dos maneras de llevar a cabo la instalación: de manera simplificada o paso a paso.

- *De manera simplificada*

Habrá que tener en cuenta los siguientes argumentos de entrada:

1. *--install=/path/to/release.tar*. Ruta donde se encuentra el archivo release.tar. En este caso */root*
2. *–install-dir=directory.* Ruta donde se quiere instalar el directorio condor. En este caso */usr/local/condor*
3. *–local-dir==directory.* Ruta del directorio local. En este caso */home/condor*
4. *–type=manager,execute,submit.* Esta opción los distintos papeles que la máquina desempeñará en el Pool de condor: Central Manager, Submit o Execute. Los distintos papales que la máquina desempeñe irán separados por comas.

Hay que configurar condor primero en la máquina que funcione como Central Manager. Cuando se tenga en claro como se quiere configurar la máquina en cuestión, habrá que ejecutar *condor_configure* seguido de las distintas opciones que se deseen.

- *Paso a paso*

Esta fue la opción por la que se optó en la instalación del Pool. Para iniciar la instalación se ejecutará desde el directorio */root/condor-6.8.4/* el comando:

perl condor_install

Una vez hecho esto comenzará una instalación guiada, que queda descrita en los siguientes pasos.

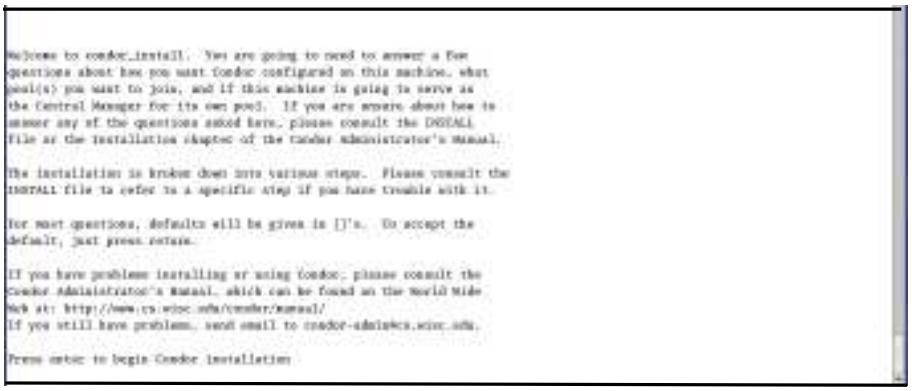


Figura 3. 19 Inicio de la instalación mediante condor_install

Paso1: ¿Qué tipo de instalación de condor se desea?

Hay tres tipos de instalación para elegir: ‘submit-only’,’full-install’, y ‘central manager‘. Una máquina Submit Only puede enviar tareas al Pool, pero no podrán

ejecutarse tareas de condor en dicha máquina. Una máquina Full-Install puede lanzar y ejecutar tareas.

Si se quiere ejecutar tareas de Condor, se debe instalar y ejecutar Condor como Root o como usuario condor.

Si lo que se desea es instalar una máquina del tipo Submit Only, se puede instalar Condor como Root o como usuario condor, o se puede instalar Condor por sí mismo en el directorio */home*.

La otra posible instalación es la creación de una máquina que actúe como Central Manager. Si se hace una instalación del tipo *full-install*, y se indica que se desea que el local host sea el Central Manager, este paso lo hará de manera automática. Sólo se debe elegir la opción *central manager* en el paso 1 si se ha ejecutado "condor_install" en el servidor de ficheros, y ahora se quiere ejecutar "condor_install" en una máquina diferente, que debe ser el Central Manager. En este caso se ha optado por una instalación del tipo:

Máquina	Papel	Tipo de instalación
1ª	Central Manager, Submit, Execute	full-install
2ª	Submit, Execute	full-install
3ª	Submit, Execute	full-install
4ª	Submit	submit-only

Tabla 3. 5 Tipo de instalación de las máquinas de la red prototipo

Paso2: ¿Cuántas máquinas se quieren establecer con este tipo de instalación?

Si se desea instalar Condor en múltiples máquinas, y se posee un sistema de archivos compartido, entonces Condor preguntará por el hostname de cada una de las máquinas que se desea añadir al Pool. Si no se posee un sistema de archivos compartido, se debe ejecutar condor_install en cada una de las máquinas donde se desea instalar Condor, por lo que Condor no preguntará por los nombres de las máquinas. Si se le provee de dicha lista, el programa de instalación creará en todas las máquinas los directorios y ficheros necesarios. Al final Condor volcará esta lista a un fichero *roster* el cual puede ser usado por los scripts para ayudar a mantener el Pool.

En este caso, se desea que se puedan añadir máquinas al Pool de manera dinámica, por lo que, a pesar de poseer un sistema de ficheros compartido, no se puede dar una lista de las máquinas fijas que usarán el Pool. Por ello debe de contestarse que no a esa pregunta.

Paso 3: Instalar el directorio Condor /release

El directorio */release* contiene 5 subdirectorios: */bin, /etc, /lib, /libexec* y */sbin.*

- */bin* contiene programas ejecutables a nivel de usuario
- */etc* es la localización recomendada para los ficheros de configuración de Condor, y a su vez posee un subdirectorio */examples*, con los ficheros de configuración por defecto, y otros ficheros empleados para la instalación de Condor.

- */lib* contiene librerías para enlazar los programas de usuario Condor, y los scripts usados por Condor.
- */sbin* contiene todos los programas administrativos ejecutables, y los demonios de Condor.
- */libexec* contiene programas que sólo Condor necesita ejecutar.

Si se poseen varias máquinas con un sistema de archivos compartido, que deben ejecutar Condor, hay que poner el directorio *release* en dicho sistema de ficheros, de modo que sólo se posea una copia de todos los binarios, de modo que cuando sea necesario actualizarlos, sólo será necesario hacerlo en un lugar. Hay que tener en cuenta que el directorio release depende de la arquitectura, por lo que a la hora de descargarse los binarios, hay que tener en cuenta la arquitectura de cada una de las máquinas del Pool.

condor_install intentará de encontrar algún directorio release instalado anteriormente:

- Si no puede encontrar ninguno, preguntará si se ha instalado alguno anteriormente. Si no se ha instalado ninguno, el programa intentará instalarlo por sí mismo descomprimiendo el directorio release (que quizá no se descomprimió anteriormente)
- Si ya se ha instalado el directorio release, preguntará si la ruta donde se encuentra instalado, es el que se desea para la instalación. Si no es así debe indicarse la ruta que se desea.

Se ha optado en este caso por la ruta de instalación */usr/local/condor* (que es la ruta por defecto), para que el programa descomprima el release.tar.

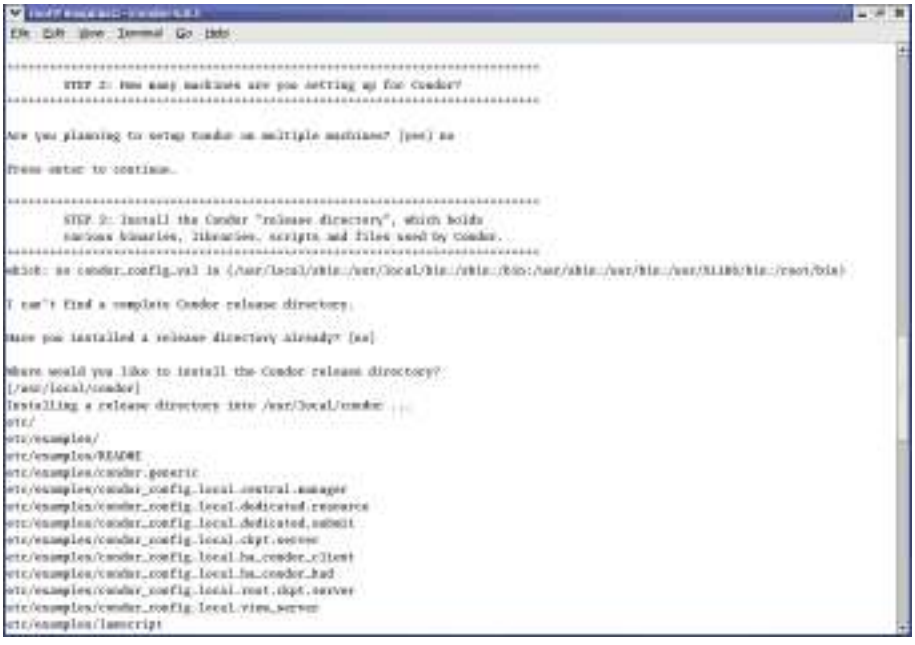


Figura 3. 20 Descomprimir el archivo release.tar

Si se desea descomprimir el archivo release.tar de manera manual, sólo habrá que ejecutar en */root/condor-6.8.5:*

tar –xvf release.tar.

Si lo que desea es creación de un Central Manager (se debe haber elegido la opción Central Manager en el paso 1), el paso número 3 es la última pregunta que se necesita responder.

Paso 4: ¿Dónde debe Condor mandar un e-mail si algo va mal?

Varias partes de Condor, mandarán un correo al administrador de Condor si algo va mal y necesita de atención humana para solucionarlo. Para ello se necesita especificar la dirección e-mail del administrador.

También se necesita especificar la ruta completa para el programa “mail” que Condor usa para enviar el e-mail. El programa mail necesita comprender la opción –s que Condor emplea para enviar el e-mail. La opcion por defecto que aparece será probablemente la correcta. En las máquinas Linux, como hay muchas distribuciones e instalaciones distintas, hay que verificar cual es la que usa por defecto. Si el programa de instalación parece no encontrar el programa mail que está especificado, habrá que buscar la ruta del programa con el comando:

wich mail

Si no se consigue encontrar la ruta se puede probar con:

wich mailx

Si aun así no se puede encontrar, deberá consultarse al administrador del sistema. Para verificar si admite la opción –s se puede consultar la página man.

En este Pool se mandarán correos al Central Manager (que actúa como administrador del sistema), estando el programa mail en */bin/mail* (ruta por defecto).

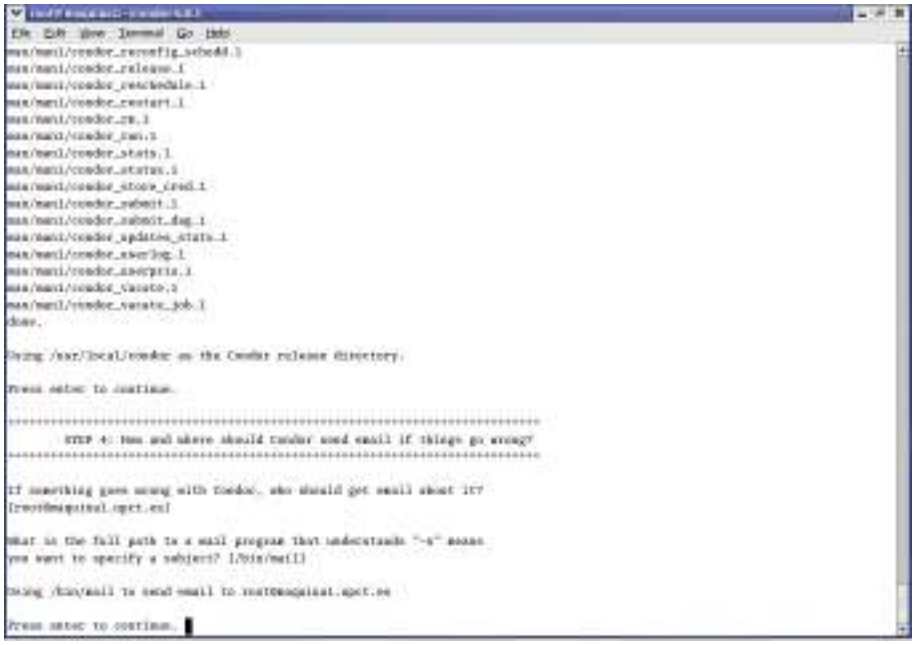


Figura 3. 21 Dirección del administrador en Redhat

Paso 5: Sistema de ficheros y dominios UID

Condor no tiene dependencia de un sistema de ficheros y un espacio UID común fuera del Universo Standard. Los trabajos pueden especificar el fichero de transferencia en al archivo submit. Para tareas del Universo Standard si será necesario. Para usar las ventajas de un sistema de archivos (como por ejemplo NFS), se necesita un UID común. Es importante configurar condor de manera correcta en un sistema de ficheros distribuido. Se preguntará si se posee un sistema de ficheros distribuido. Si es así, condor_install configurará el FILESYSTEM_DOMAIN para ser el dominio de la máquina donde se está ejecutando *condor_install*. Si no es así se deberá poner:

FILESYSTEM_DOMAIN=$(FULLHOSTNAME)

indicando que cada máquina posee su propio dominio. Para el dominio UID Condor debe saber si todas las máquinas que forman parte del Pool poseen el mismo UID. Si es así, UID debe ser establecido como el dominio propio de cada máquina. Si no es así se establecerá como:

UID_DOMAIN=$(FULLHOSTNAME)

indicando que cada máquina posee su propio dominio. Si se posee un UID común, *condor_install* preguntará si se trata de un dominio UID "soft", significando, que aunque se tenga un único UID, no todas las máquinas en el Pool, tienen a todos los usuarios en ficheros password individuales.

En la solución dada para este Pool en concreto, se ha optado por un sistemas de ficheros compartido en el dominio upct.es, un UID único (upct.es), por si se quisieran ejecutar tareas dentro del Universo Standard, y todos los usuarios tendrán ficheros password individuales.

Paso 6: Soporte al Universo Java en Condor

Condor tiene la posibilidad de ejecutar programas de Java con entrada y salida remota, pero sin checkpointing. Si se quiere habilitar esta posibilidad en Condor, habrá que indicar que si se quiere establecer el Universo Java en la máquina. El instalador intentará establecer si se posee una JVM (Java Virtual Machine) válida y si no encuentra ninguna se lo indicará al usuario. Si no se desea ejecutar el Universo Java (como es este caso), entonces es más seguro contestar que no en esta parte.

Paso 7: ¿Donde deben instalarse los programas públicos?

Es recomendable que los programas de Condor (a nivel de usuario se instalen en el directorio release). De esta manera, cuando se quiera instalar una nueva versión de los binarios de Condor, sólo será necesario reemplazar el directorio release y todo será actualizado. Por eso una opción es tener a los usuarios condor en *<release_dir>/bin*. Sin embargo la opción que se recomienda, y que se ha empleado para este Pool, es crear enlaces simbólicos al directorio */usr/local/bin*, y que desde allí se llame a los programas de Condor. *condor_install* hace esto si se desea. Lo único que debe hacerse es indicarle hacia que directorio quieres hacer el enlace. De esta manera los usuarios no tienen que cambiar su path para usar Condor, y se pueden tener los binarios instalados en su propia localización.

Si no se está instalando Condor ni como Root, ni como usuario Condor hay un script perl, que funciona de envoltorio a todas las herramientas de Condor que se crean, establece las variables de entorno adecuadas, y automáticamente le pasa ciertas opciones a las herramientas. Todo esto se hace de manera automática por condor_install, para lo cual se necesita indicarle a Condor donde pone el script perl. El script por sí mismo. Este paso puede observarse en la Figura 3.3.4.7. A partir de es donde se diferencian las instalaciones:

❖ Instalación Full Install

Paso 8: ¿Qué máquina va a ser el Central Manager?

Hay que escribir el hostname completo de la máquina que se ha seleccionado para que sea el Central Manager. Si *condor_install* no puede encontrar información sobre el host que se le ha indicado, mostrará un mensaje de error y te pedirá que se solucione el problema.

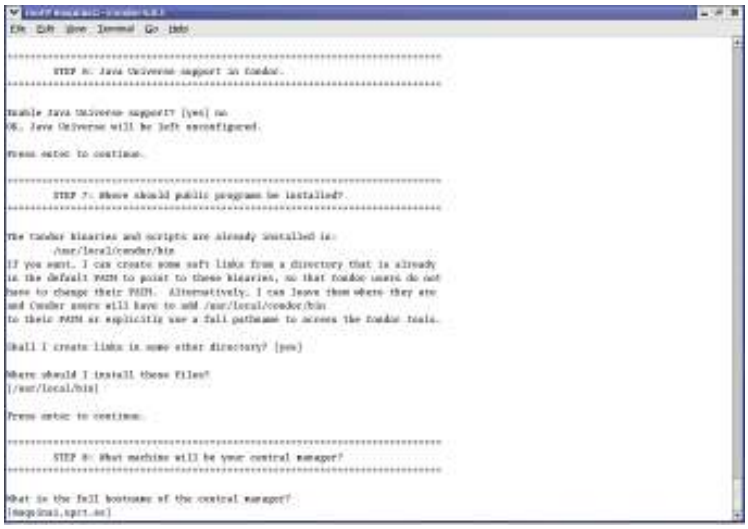


Figura 3. 22 Instalación de los programas públicos y elección del Central Manager

En el Pool de este proyecto el Central Manager es la máquina 192.168.1.1 (maquina1.upct.es), que es el hostname que se le pasa en este paso.

Paso 9: ¿Dónde debe situarse el directorio local?

Este es el directorio que se comentó en el quinto punto del apartado 3.2.4. *condor_install* trata de hacer una serie de conjeturas sobre el directorio que se desea usar para dicho propósito. Se puede contestar que sí cuanto te pregunte por el directorio que se desea, o bien escribirlo. Como el directorio debe de ser único, es normal usar el hostname de cada máquina. Cuando se está escribiendo en el propio path, se puede emplear '$(HOSTNAME)', que Condor tomará como el hostname de la máquina en la que se ejecuta. A partir de este momento Condor intentará crear los directorios necesarios en todas las máquinas que se indicaron en el Paso 2.

Una vez que se haya seleccionado el directorio local, *condor_install* creará los subdirectorios necesarios con los permisos adecuados. Deben de tener los siguientes permisos y propiedades:

```
dwxr-xr-x    2 condor   root 1024 Sep     6 15:35 execute/
dwxr-xr-x      2 condor   root       1024 Sep     6 15:35 log/
dwxr-xr-x      2 condor   root       1024 Sep     6 15:35 sPool/
```

En este proyecto el directorio local se ha situado en */usr/local/condor/home.*Si el directorio local se encuentra en un sistema de archivos compartido Condor te preguntará por la localización de los archivos lock. En este caso, cuando condor_install termina, hay que ejecutar condor_init en cada máquina del Pool para crear el directorio lock antes de lanzar Condor.

Paso 10: ¿Dónde deben situarse los ficheros de configuración local?

Hay diferentes niveles dentro de los ficheros de configuración. Hay un fichero de configuración local que debe instalarse en */usr/local/condor/etc/condor_config*, y hay otros que son específicos de la máquina, que pueden sobrescribir características del fichero de configuración global. Si se ha instalado Condor en múltiples máquinas o se está configurando la máquina que actúa como Central Manager, se debe elegir la ruta de instalación de los ficheros de configuración locales. as dos opciones son:

- Tener un solo directorio que posea todos los ficheros de configuración local que se llame $(HOSTNAME).local. Esto se recomienda cuando se tiene un sistema de archivos distribuido, ya que es más sencillo configurar un Pool desde un ligar centralizado

- Tener ficheros de configuración independientes en los directorios locales de cada máquina

Debido a que no se puede establecer con exactitud el número de máquinas que van a formar parte del Pool (nuevas máquinas se incorporan para lanzar tareas, se añaden nuevas máquinas en los laboratorios), se ha optado por la segunda opción, situándose el fichero de configuración local en /usr/local/condor/etc/$(HOSTNAME).local. Cuando se ha hecho este paso, el programa de instalación, le indica al archivo de configuración global la situación del fichero de configuración local. demás si se trata de la máquina que actúa como Central Manager (como es el caso de la 3.3.4.9) éste será el momento en el que se active la máquina como Central Manager.

Una vez hecho esto, se pedirá (en el caso de que sea el Central Manager) el nombre del Pool. En este caso en particular se optó por **PROYECTO-UPCT-CONDOR.**

Paso 11: ¿Dónde debe Condor buscar estos ficheros de configuración?

Hay una serie de lugares conocidos, donde Condor busca los ficheros de configuración, por lo que es recomendable hacer enlaces desde alguno de ellos hacia */usr/local/condor/ect/condor_config*. De esa manera se puede tener guarda la configuración de Condor en un lugar centralizado, pero todos los demonios de Condor serán capaces de encontrar sus ficheros de configuración. También se le puede dar a la variable de entorno CONDOR_CONFIG el valor */usr/local/condor/etc/condor_config*.

Una vez que se completa este paso de la instalación full_install, condor_install muestra mensajes describiendo que hay que hacer una vez que se termina la instalación. n el Pool del proyecto se ha elegido este tipo de instalación para la mayor parte de las máquinas. La idea es que los ordenadores que vayan a ejecutar tareas (es decir, los que formen parte del laboratorio) tengan este tipo de instalación, con un Central Manager por laboratorio.

❖ Instalación Submit Only

Esta instalación implica que la máquina podrá lanzar tareas de Condor en uno o más Pools. Para este tipo de instalación es necesario que la máquina se subscriba a dichos Pools. ra instalarlo hay que volver al paso 6 y continuar.

Paso 6: continuación

Se tiene la opción de lanzar tareas a más de un Pool. Para poder inscribirse a un Pool, es necesario indicar el hostname de cada máquina del Pool. El primer Pool que se indique será el que se tome por defecto para comenzar y lanzar tareas. ay un fichero de configuración para cada Pool. La localización de cada fichero debe especificarse.

La identificación de cada Pool requiere un único nombre. Un último apartado relaciona cada Pool con un nombre. El nombre debe ser el argumento para la opción **–Pool** de la línea de comandos.

En el Pool del proyecto, sólo se ha llevado a cabo esta instalación el la máquina 4ª, pero la idea es que los usuarios externos lleven a cabo esta instalación, de modo que sólo tengan que lanzar tareas al Pool

Una vez que Condor ha sido instalado en las máquinas que van a formar parte del Pool, hay algunas cosas que deben comprobarse antes de realizar la instalación.

1) Hay que mirar el fichero de configuración global (/usr/local/condor/etc/condor_config). Hay algunas características a las

que se debe echar un vistazo, para asegurarse de que el Pool funciona como debe.

2) Condor puede monitorizar la actividad del teclado y la del ratón. Esto puede indicarse, modificando la variable CONSOLE_DEVICES de la sección condor_startd del archivo de configuración. Esta variable es una lista separada por comas, por lo que se puede tener cualquier dispositivo en el directorio */dev* como 'console devices' cuya actividad le será mandada al demonio **condor_startd** como *ConsoleIdleTime*.

3) (Sólo para Linux). Condor necesita ser capaz de encontrar el fichero utmp. Este fichero debe encontrarse en el directorio /var/run/utmp. Si no lo encuentra en dicho directorio, lo buscará en /var/adm/utmp. Si aún así no lo encuentra se da por vencido. Por ello si está en otro lugar se deben crear enlaces simbólicos desde donde se sitúen hacia */var/run/utmp*.

Lo siguiente que se debe hacer es ejecutar **condor_init** en todas las máquinas que van a formar parte del Pool para crear los directorios LOCK , ya que en este caso no han sido creados por el programa condor_install. Además con el comando condor_init se asegura que han sido creados todos los directorios que serán imprescindibles en Condor (/log, /sPool, /execute) así como también el fichero de configuración local, y el fichero de configuración global. Para ello es necesario situarse en */usr/local/condor/sbin/* y ejecutar **./condor_init**

Figura 3. 23 Fin de la instalación

Además de esto si se instala Condor en un sistema de ficheros distribuido, condor_init crea enlaces hacia cada una de las máquinas que van a formar parte del Pool y se indican en el fichero de configuración. Estos enlaces se hacen para que las máquinas puedan encontrar el fichero de configuración global, pero este no el caso de esta instalación. ambién debe copiarse el script *condor_generic* de la ruta */usr/local/etc/examples/condor_generic* a */usr/local/bin/condor.* Una vez copiado, deben de realizarse los siguientes cambios:

```
#!/usr/bin/env perl


$default_Pool="default";
$release_dir="/usr/local/condor";
$sbin="$release_dir/sbin";
$bin="$release_dir/bin";
%configlocation = (
                "default","/usr/local/condor/etc/condor_config"
);
```

Para que Condor se inicie cada vez que arranque la máquina, se debe copiar el script *condor_boot,* localizado en la ruta */usr/local/condor/etc/examples/condor.boot* en la siguiente ruta

[condor@maquina1 examples]$*cp –p /uss/local/condor/etc/examples/condor.boot /etc/init.d/condor*

Se debe crear un enlace simbólico entre los ficheros que se citan a continuación:

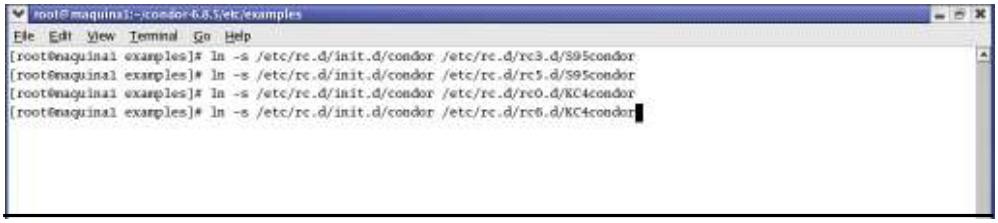


Figura 3. 24 Enlaces simbólicos para el inicio de Condor

El comando *ln* crea un enlace entre un fichero existente y un nombre nuevo de fichero. Ambos son idénticos para el usuario. La opción *–s* realiza un enlace simbólico a pesar del enlace existente. ara una completa instalación del script **condor_compile** se debe cambiar el nombre el enlazador ld de la ruta */usr/bin* por ld.real, y copiar el enlazador de Condor en dicha ruta:

[condor@maquina1 bin]$ *mv /usr/bin/ld /usr/bin/ld.real*

[condor@maquina1 bin]$ *cp –p /usr/local/condor/lib/ld /usr/bin/ld*.

El siguiente paso, como ya se indicó en el punto 1 es la modificación de los archivos de configuración:

- **Fichero de configuración global**

Las modificaciones realizadas en el archivo de configuración global han sido señalizadas, así como aquellos parámetros que se consideran interesantes. Gan parte de los parámetros, han sido establecidos por defecto gracias al script condor_install.

Se han eliminado aquellas partes que no son importantes en una instalación básica, y aquéllas que serán utilizadas para funciones específicas más adelante.

En este Pool en específico fue fundamental el cambio del parámetro *RunBenchmarks* a False, ya que debido a las características especiales de las máquinas del laboratorio, este parámetro no funcionaba bien, e impedía que las máquinas pasaran del estado *Owner* al de *Unclaimed* y por lo tanto hacía imposible que las máquinas pudieran ejecutar tareas.

[condor@maquina1 etc]$ /usr/local/condor/etc/ emacs condor_config

```
##  Part 1:  Settings you must customize:
##########################################################################
##########################################################################
LOCAL_DIR          = $(RELEASE_DIR)/home
#LOCAL_DIR         = $(RELEASE_DIR)/hosts/$(HOSTNAME)

#LOCAL_CONFIG_FILE      = $(LOCAL_DIR)/condor_config.local
LOCAL_CONFIG_FILE = $(RELEASE_DIR)/etc/$(HOSTNAME).local

UID_DOMAIN  = upct.es

FILESYSTEM_DOMAIN = upct.es
```

```
COLLECTOR_NAME          = PROYECTO-UPCT-CONDOR

##--------------------------------------------------------------------
##  Part 2:  Settings you may want to customize:
##  (it is generally safe to leave these untouched)
##--------------------------------------------------------------------
##--------------------------------------------------------------------

CONDOR_IDS=0.0

HOSTALLOW_ADMINISTRATOR = $(CONDOR_HOST)

HOSTALLOW_OWNER = $(FULL_HOSTNAME), $(HOSTALLOW_ADMINISTRATOR)

HOSTALLOW_READ =192.168.1.*

HOSTALLOW_WRITE = 192.168.1.*

HOSTALLOW_NEGOTIATOR = $(CONDOR_HOST)

HOSTALLOW_WRITE_COLLECTOR = $(HOSTALLOW_WRITE), $(FLOCK_FROM)
HOSTALLOW_WRITE_STARTD    = $(HOSTALLOW_WRITE), $(FLOCK_FROM)
HOSTALLOW_READ_COLLECTOR  = $(HOSTALLOW_READ), $(FLOCK_FROM)
HOSTALLOW_READ_STARTD     = $(HOSTALLOW_READ), $(FLOCK_FROM)

USE_NFS           = True


USE_AFS           = False

MAX_JOBS_RUNNING  = 200

LOCK        = /var/lock/condor

CREATE_CORE_FILES = False

##--------------------------------------------------------------------
##  Part 3:  Settings control the policy for running, stopping, and
##  periodically checkpointing condor jobs:
##--------------------------------------------------------------------

MINUTE            = 60
HOUR        = (60 * $(MINUTE))
StateTimer  = (CurrentTime - EnteredCurrentState)
ActivityTimer     = (CurrentTime - EnteredCurrentActivity)
ActivationTimer = (CurrentTime - JobStart)
LastCkpt    = (CurrentTime - LastPeriodicCheckpoint)

STANDARD    = 1
PVM         = 4
VANILLA     = 5
MPI         = 8
IsPVM       = (TARGET.JobUniverse == $(PVM))
IsMPI       = (TARGET.JobUniverse == $(MPI))
IsVanilla   = (TARGET.JobUniverse == $(VANILLA))
IsStandard      = (TARGET.JobUniverse == $(STANDARD))

NonCondorLoadAvg  = (LoadAvg - CondorLoadAvg)
BackgroundLoad    = 0.3
HighLoad          = 0.5
StartIdleTime     =  5 * $(MINUTE)
```

```
ContinueIdleTime  =  5 * $(MINUTE)
MaxSuspendTime    = 10 * $(MINUTE)
MaxVacateTime     = 10 * $(MINUTE)

KeyboardBusy           = (KeyboardIdle < $(MINUTE))
ConsoleBusy            = (ConsoleIdle  < $(MINUTE))
CPUIdle                = ($(NonCondorLoadAvg) <= $(BackgroundLoad))
CPUBusy                = ($(NonCondorLoadAvg) >= $(HighLoad))
KeyboardNotBusy        = ($(KeyboardBusy) == False)

BigJob   =(TARGET.ImageSize >= (50 * 1024))
MediumJob=(TARGET.ImageSize    >=    (15    *    1024)&&TARGET.ImageSize
<(50*1024))
SmallJob    = (TARGET.ImageSize <  (15 * 1024))

JustCPU          = ($(CPUBusy) && ($(KeyboardBusy) == False))
MachineBusy      = ($(CPUBusy) || $(KeyboardBusy))

RANK             = 0

TESTINGMODE_WANT_SUSPEND      = False
TESTINGMODE_WANT_VACATE       = False
TESTINGMODE_START             = True
TESTINGMODE_SUSPEND           = True
TESTINGMODE_CONTINUE          = True
TESTINGMODE_PREEMPT           = True
TESTINGMODE_KILL              = False
TESTINGMODE_PERIODIC_CHECKPOINT     =True
TESTINGMODE_PREEMPTION_REQUIREMENTS =True
TESTINGMODE_PREEMPTION_RANK = 0

##########################################################################
##  Part 4:  Settings you should probably leave alone:
##  (unless you know what you're doing)
##########################################################################
##########################################################################

LOG         = $(LOCAL_DIR)/log
SPOOL       = $(LOCAL_DIR)/sPool
EXECUTE     = $(LOCAL_DIR)/execute
BIN         = $(RELEASE_DIR)/bin
LIB         = $(RELEASE_DIR)/lib
INCLUDE     = $(RELEASE_DIR)/include
SBIN        = $(RELEASE_DIR)/sbin
LIBEXEC     = $(RELEASE_DIR)/libexec

HISTORY           = $(SPOOL)/history

COLLECTOR_LOG     = $(LOG)/CollectorLog
KBDD_LOG          = $(LOG)/KbdLog
MASTER_LOG        = $(LOG)/MasterLog
NEGOTIATOR_LOG    = $(LOG)/NegotiatorLog
NEGOTIATOR_MATCH_LOG = $(LOG)/MatchLog
SCHEDD_LOG  = $(LOG)/SchedLog
SHADOW_LOG  = $(LOG)/ShadowLog
STARTD_LOG  = $(LOG)/StartLog
STARTER_LOG = $(LOG)/StarterLog

SHADOW_LOCK = $(LOCK)/ShadowLock

COLLECTOR_HOST  = $(CONDOR_HOST)
```

```
NEGOTIATOR_HOST = $(CONDOR_HOST)

SHUTDOWN_GRACEFUL_TIMEOUT     = 1800

RESERVED_DISK           = 5

RESERVE_AFS_CACHE = False


##  Daemon-specific settings:

##  condor_master

MASTER                  = $(SBIN)/condor_master
STARTD                  = $(SBIN)/condor_startd
SCHEDD                  = $(SBIN)/condor_schedd
KBDD                    = $(SBIN)/condor_kbdd
NEGOTIATOR              = $(SBIN)/condor_negotiator
COLLECTOR               = $(SBIN)/condor_collector
STARTER_LOCAL           = $(SBIN)/condor_starter

MASTER_ADDRESS_FILE = $(LOG)/.master_address

PREEN             = $(SBIN)/condor_preen

PREEN_ARGS              = -m -r

PREEN_INTERVAL                  = 86400

PUBLISH_OBITUARIES              = True

OBITUARY_LOG_LENGTH             = 20

START_MASTER                    = True

START_DAEMONS                   = True

MASTER_UPDATE_INTERVAL          = 300

MASTER_CHECK_NEW_EXEC_INTERVAL      = 300

MASTER_NEW_BINARY_DELAY = 120

SHUTDOWN_FAST_TIMEOUT           = 120

:
MASTER_BACKOFF_FACTOR           = 2.0


MASTER_BACKOFF_CEILING          = 3600

MASTER_RECOVER_FACTOR           = 300

##  condor_startd
```

```
STARTER_LIST      = STARTER, STARTER_PVM, STARTER_STANDARD
STARTER           = $(SBIN)/condor_starter
STARTER_PVM       = $(SBIN)/condor_starter.pvm
STARTER_STANDARD  = $(SBIN)/condor_starter.std
STARTER_LOCAL     = $(SBIN)/condor_starter

STARTD_ADDRESS_FILE     = $(LOG)/.startd_address

POLLING_INTERVAL         = 5

UPDATE_INTERVAL          = 300

MATCH_TIMEOUT            = 300

KILLING_TIMEOUT   = 30

RunBenchmarks : False

STARTD_HAS_BAD_UTMP = True

CONSOLE_DEVICES   = mouse, console

MY_CONSOLE_DEVICES = "$(CONSOLE_DEVICES)"

STARTD_ATTRS = MY_CONSOLE_DEVICES, RunBenchmarks, UPDATE_INTERVAL

##--------------------------------------------------------------------
##  condor_schedd
##--------------------------------------------------------------------

SHADOW_LIST = SHADOW, SHADOW_PVM, SHADOW_STANDARD
SHADOW            = $(SBIN)/condor_shadow
SHADOW_PVM        = $(SBIN)/condor_shadow.pvm
SHADOW_STANDARD   = $(SBIN)/condor_shadow.std

SCHEDD_ADDRESS_FILE     = $(LOG)/.schedd_address

SCHEDD_INTERVAL   = 300

JOB_START_DELAY   = 2

ALIVE_INTERVAL          = 300

MAX_SHADOW_EXCEPTIONS   = 5

SHADOW_SIZE_ESTIMATE    = 1800

SHADOW_RENICE_INCREMENT = 10

NEGOTIATE_ALL_JOBS_IN_CLUSTER = True

QUEUE_SUPER_USERS = root, condor

##--------------------------------------------------------------------
##  condor_starter
##--------------------------------------------------------------------

JOB_RENICE_INCREMENT    = 10

STARTER_LOCAL_LOGGING   = TRUE
```

```
SOFT_UID_DOMAIN   = FALSE

##--------------------------------------------------------------------
##  condor_submit
##--------------------------------------------------------------------

MACHINE = "$(FULL_HOSTNAME)"
SUBMIT_EXPRS = MACHINE

DEFAULT_IO_BUFFER_SIZE = 524288

DEFAULT_IO_BUFFER_BLOCK_SIZE = 32768

##--------------------------------------------------------------------
##  condor_preen
##--------------------------------------------------------------------

PREEN_ADMIN       = $(CONDOR_ADMIN)

VALID_SPOOL_FILES =    job_queue.log,    job_queue.log.tmp,    history,
\Accountant.log,         Accountantnew.log,         \local_univ_execute,
.quillwritepassword

INVALID_LOG_FILES = core

##--------------------------------------------------------------------
##  condor_credd credential managment daemon
##--------------------------------------------------------------------

CREDD                     = $(SBIN)/condor_credd

CREDD_ADDRESS_FILE      = $(LOG)/.credd_address
```

- **Fichero de configuración local**

Una vez que se han modificado los parámetros del archivo de configuración global del Pool, debe procederse a la modificación de los archivos de configuración local del Central Manager situado en */usr/local/condor/etc/maquina1.local*. Del mismo modo que en el fichero de configuración global se han señalizado los cambios realizados, y aquellas variables que deben tenerse en cuenta.

[condor@maquina1 etc]$ *emacs maquina1.local*

Del mismo modo que en el fichero de configuración global se han señalizado los cambios realizados, y aquellas variables que deben tenerse en cuenta.

```
##########################################################################
##
##  condor_config.local.central.manager
##########################################################################

COLLECTOR_NAME          = PROYECTO-UPCT-CONDOR

DAEMON_LIST   = MASTER, COLLECTOR, NEGOTIATOR, STARTD, SCHEDD

COLLECTOR     = $(SBIN)/condor_collector
NEGOTIATOR    = $(SBIN)/condor_negotiator

##--------------------------------------------------------------------
##  condor_collector
##--------------------------------------------------------------------
```

```
CLASSAD_LIFETIME  = 900

MASTER_CHECK_INTERVAL   = 10800

CLIENT_TIMEOUT          = 30

QUERY_TIMEOUT           = 60

##--------------------------------------------------------------------
##  condor_negotiator
##--------------------------------------------------------------------

NEGOTIATOR_INTERVAL     = 300

NEGOTIATOR_TIMEOUT      = 30

PRIORITY_HALFLIFE = 86400

NICE_USER_PRIO_FACTOR = 10000000

MAX_ACCOUNTANT_DATABASE_SIZE = 1000000

REMOTE_PRIO_FACTOR = 10000

NEGOTIATOR_SOCKET_CACHE_SIZE = 16
```

3.3.5 Puesta en marcha de los demonios Condor.

Una vez terminada la instalación, deben reiniciarse las máquinas, empezando por el Central Manager, para que los cambios se hagan efectivos.

Para poner en marcha los demonios de Condor hay que ejecutar en todas las máquinas el demonio **condor_master**, empezando por el Central Manager. Dicho demonio se encuentra en la ruta */usr/local/condor/sbin/condor_master*, aunque gracias a las modificaciones que se han realizado, se puede ejecutar **condor_master** desde cualquier ruta.

[condor@maquina1 sbin]$ *condor_master*

[condor@maquina2 sbin]$ *condor_master*

[condor@maquina3 sbin]$ *condor_master*

[condor@maquina4 sbin]$ *condor_master*

La función básica del condor_master es la de asegurarse de que los demás demonios de Condor están ejecutándose. El master vigila a los demonios, los reinicia si se paran, y periódicamente comprueba si se han instalado nuevos binarios, y reinicia los demonios si dichos binarios les afectan. ara comprobar que los demonios de Condor están funcionando correctamente se ejecuta el comando que se muestra en la siguiente figura:

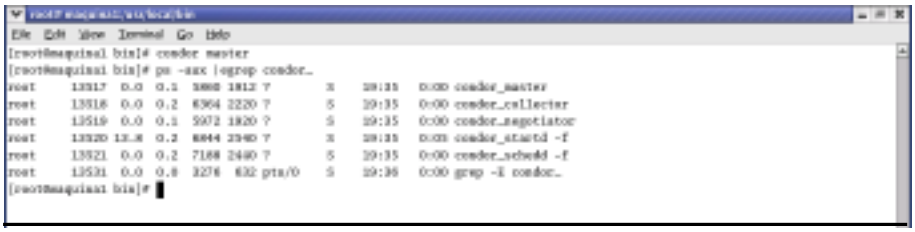


Figura 3. 25 Demonios Condor del Central Manager

En la Figura 3.3.5.1 se observan los demonios del Central Manager. os demonios de las máquinas del Pool de este proyecto son:

Máquina: Función	Demonios
1ª: Central Manager, Submit, Execute	condor_master, condor_collector, condor_negotiator, condor_startd, condor_schedd
2ª: Submit, Execute	condor_master, condor_schedd, condor_startd
3ª: Submit, Execute	condor_master, condor_schedd, condor_startd
4ª: Submit	condor_master, condor_startd

Tabla 3. 6 Demonios de las máquinas que forman parte del Pool

Una vez comprobado que los demonios funcionan correctamente, ya es posible comenzar a ejecutar comandos de Condor, y lanzar tareas.

Capítulo 4
Lanzar tareas: submitting, queueing, executing.

4.1 Fichero de descripción submit

Una tarea es lanzada para su ejecución en Condor usando el comando condor_submit. condor_submit toma como argumento el nombre de un fichero llamado fichero de descripción submit. Ese fichero contiene comandos y palabras clave para controlar el encolamiento de las tareas. En el fichero de descripción submit Condor encuentra todo lo que necesita conocer sobre la tarea. Incluye términos como el nombre del ejecutable, el directorio de trabajo inicial y argumentos de la línea de comandos del programa, todos ellos dentro del fichero de descripción submit. Con toda esta información Condor crea una ClassAd y trabaja para la ejecución de la tarea.

El contenido del fichero de descripción submit puede recortar tiempo para los usuarios de Condor. Es fácil si lo que se desea es lanzar una misma tarea muchas veces: Por ejemplo, si se ejecuta el mismo programa 500 veces con 500 entradas de datos distintas, hay que organizar los ficheros de datos de manera que cada ejecución lea su propio fichero de entrada, y que cada ejecución escriba en su propio fichero de salida. Cada ejecución individual puede tener su propio directorio de trabajo inicial, stdin, stdout, stderr, argumentos de la línea de comandos, y encapsulación. Un programa que directamente abre su propio fichero leerá los nombres de ficheros ya sea desde stdin o desde la línea de comandos. Un programa que abre un fichero estático necesitará usar un subdirectorio separado para la salida de cada ejecución.

4.1.1 ClassAd

Antes de aprender a lanzar una tarea, es importante comprender como Condor asigna recursos. Comprendiendo la única forma de asignar las tareas lanzadas con las máquinas es la clave para obtener la mejor ejecución posible.

Condor simplifica el lanzamiento de la tarea actuando el manejador de ClassAds. Las máquinas indican que atributos poseen, esperando atraer a una tarea como por ejemplo, cantidad de memoria RAM, tipo y velocidad de la CPU, memoria virtual, carga media, etc. Los que lanzan las tareas indican especificaciones sobre lo que quieren que cumpla una máquina para poder ejecutar su tarea.

El ClassAd de la máquina también indica sobre que condiciones desea ejecutar una tarea, y que tipo de tareas prefiere. Estos atributos de las políticas pueden reflejar las preferencias individuales por las cuales los diferentes propietarios han permitido que sus máquinas formen parte del Pool. Debe advertirse que la máquina sólo puede ejecutar tareas mientras que no haya actividad en la máquina. También puede indicarse una preferencia para ejecutar tareas lanzadas por uno mismo o por otra máquina.

Del mismo modo, cuando se lanza una tarea, se especifica una ClassAd con sus requisitos y preferencias. La ClassAd incluye el tipo de máquina que se desea

usar. Por ejemplo, quizás se esté buscando la representación disponible en punto flotante más rápida. Puede desearse mostrar el número de máquinas disponibles basadas en una representación en punto flotante. O quizás puede desearse que la máquina tenga un mínimo de 128 Mbytes de memoria. O puede que sólo se quiera tomar la primera de las máquinas que se posea. Estos atributos y requisitos de la tarea se unen dentro de la ClassAd de una tarea.

Condor juega el papel de manejador, leyendo de manera continua todos los ClassAds de las tareas, uniendo y organizando las tareas con las máquinas. Condor se asegura de que todos los requisitos de las ClassAds se satisfacen.

Para poder mostrar las ClassAds que hay en Condor, se deben emplear los siguientes parámetros:

Comando	Descripción
Condor_status – available	Muestra sólo máquinas que están dispuestas a ejecutar tareas ahora
Condor_status –run	Muestra sólo máquinas que están ejecutando tareas
Condor_status –l	Hacen un listado de las ClassAds de todas las máquinas del Pool.

Tabla 4. 1 Comandos para visualizar los ClassAd

4.1.2 Requirements & Rank

Los comandos de Requirements y rank del fichero de descripción submit son poderosos y flexibles. Ambos necesitan ser especificados como expresiones válidas para las ClassAds de Condor, sin embargo hay valores por defecto que son dados por el programa condor_submit si no se definen en el fichero de descripción submit.

Los operadores de comparación (<, >, <=, >=, y ==) comparan cadenas de caracteres. Los operadores de comparación especiales =?= y =!= hacen comparaciones especiales.

Los atributos de las ClassAds permitidos son aquellos que forman parte de una máquina o de la ClassAd de una tarea. Para poder visualizar todos los atributos ClassAd de todas las máquinas del Pool de Condor, hay que ejecutar condor_status –l. El argumento –l de condor_status indica que hay que mostrar todos los atributos ClassAd de todas las máquinas. Los atributos ClassAd de las tareas que hay en la cola se visualizan mediante condor_q –l. El conjunto de ambas todos los atributos permitidos con los que se cuenta.

Para ayudar a entender cuales de estos atributos son significativos, los descriptores buscan los atributos que son comunes para todos los ClassAd de las máquinas. Hay que recordar que debido a que los ClassAd son flexibles, las máquinas incluirán atributos específicos adicionales para la instalación y las políticas de la máquina. A continuación se muestra una tabla con los ClassAds más significativos:

Atributo ClassAd	Descripción
Activity	Cadena que describe la actividad de trabajo de Condor en la máquina. Puede tener uno de los siguientes valores:Idle, Busy, Suspended, Vacating, Killing, Benchmarking.
Arch	Cadena que indica la arquitectura de la máquina. En el caso de este proyecto se empleó la cadena “INTEL” que describe a un Intel x86.
Console Idle	Número de segundos transcurridos desde que se detectó actividad de la consola (ya sea ratón o teclado).

FileSystemDomain	Nombre de dominio configurado por el administrador de Condor que describe a un clúster de máquinas en el que todas las máquinas acceden de la misma manera uniforme, ya sea vía NFS o AFS. Es útil para las tareas Vanilla con acceso a ficheros remotos.
Machina	Una cadena que define el hostname de la máquina.
OpSys	Cadena que describe el sistema operativo de la máquina. En este proyecto se han empleado “LINUX” para Redhat y “WINNT51” para Windows XP.
Requirements	Un booleano, que cuando se evalúa a través del contexto ClassAd o de una tarea ClassAd, debe ser evaluado a TRUE antes de que Condor pueda usar la máquina.
JobId	El identificador de la tarea, que se puede observar por condor_q en la máquina donde se lanzó la tarea.
Cmd	La ruta y el nombre del fichero donde se debe ejecutar la tarea.
ImageSize	Tamaño de la imagen de la tarea en Kbytes de memoria.
JobUniverse	Entero que indica el universo donde ejecutar la tarea.
Owner	Cadena que describe el usuario que lanzó la tarea.
TransferIn	Si es True, entonces la entrada de la tarea se transfiere desde la máquina que lanzó la tarea a la máquina remota. El nombre del fichero que se transfiere se da en el atributo In.
TransferOut	Si es true, entonces la salida de la tarea se transfiere desde la máquina remota de regreso a la máquina que lanzó la tarea. El nombre del fichero a transmitir se da en el atributo Out.

Tabla 4. 2 Descripción de los atributos más significativos de los ClassAd

Expresiones Rank

Cuando se consideran las relaciones entre una tarea y una máquina, la expresión Rank se emplea para elegir una de las máquinas entre todas las existentes que satisfaga las necesidades de la tarea, y estén disponibles para el usuario, teniendo en cuenta las prioridades y el rango de la tarea. Las expresiones Rank, simples o complejas, definen un valor numérico que expresa preferencias.

La expresión Rank de la tarea se evalúa a uno de estos tres valores: UNDEFINED, ERROR, o un valor en punto flotante. Si es un valor en punto flotante, la mejor elección será aquella con el mayor valor positivo. Si no se da ninguna expresión Rank, entonces Condor sustituye el valor por defecto por 0.0. Si adquiere el valor UNDEFINED o ERROR, también se usa el valor 0.0. Por lo tanto, a la máquina se la sigue considerando como válida para la asignación de la tarea, pero no hay ningún valor rank que considerar por encima de otro. Un ejemplo de este tipo de expresión sería:

```
Rank=HAS_MATLAB
```

En este caso Condor sólo buscará aquellas máquinas que posean Matlab.

4.1.3 Lanzar tareas empleando o no un Sistema de ficheros Distribuido

Caso RedHat: Uso de un sistema de ficheros distribuido.

Si las tareas de los universos Vanilla, Java, Paralelo (o MPI) se lanzan sin usar un mecanismo de transmisión de ficheros, Condor debe usar un sistema de ficheros

distribuidos para acceder a los ficheros de entrada y salida. En este caso la máquina debe de ser capaz de acceder a los ficheros de información desde cualquier máquina en la que pueda ejecutarse de manera potencial.

Condor permite a los usuarios asegurarse de que sus tareas tienen acceso al sistema de ficheros, usando los atributos ClassAds FileSystemDomain y UidDomain. Este atributo especifica que máquinas poseen acceso al mismo sistema de ficheros. Todas las máquinas que poseen los mismos directorios compartidos se consideran pertenecientes al mismo sistema de ficheros. De manera similar, todas las máquinas que poseen la misma información de usuario son considerados parte del mismo dominio UID. Por ello si un Pool debe tener acceso a un sistema de ficheros distribuido, el administrador del Pool deben de configurar de manera correcta Condor de forma que todas las máquinas que comparten los mismos ficheros posean el mismo FileSystemDomain en el archivo de configuración. De la misma manera todas las máquinas que posean información común de usuario debe ser configurado con el mismo UidDomain del archivo de configuración.

Cuando una máquina depende de un sistema de ficheros distribuido, Condor emplea la expresión Requirements para asegurarse de que la máquina se ejecuta en el correcto UidDomain y el correcto FileSystemdomain. En este caso la expresión Requirements por defecto especifica que la tarea debe ejecutarse en una máquina con el mismo UidDomain y FileSystemDomain de la máquina donde se lanzó la tarea. Este valor por defecto es casi siempre el correcto. Sin embargo en un Pool con múltiples UidDomain y FileSystemDomain, el usuario necesita especificar una expresión Requirements para poder ejecutar las tareas en las máquinas adecuadas.

Caso Windows: Lanzar tareas sin un sistema de archivos distribuido. Empleo del mecanismo de transferencia de ficheros.

Este mecanismo lo emplea el usuario cuando éste lanza una tarea. Condor transferirá todos los ficheros necesarios para la ejecución de una tarea desde la máquina donde la tarea fue lanzada, a un directorio de trabajo temporal en la máquina donde se ejecuta la tarea. Condor ejecuta la tarea y transfiere la salida de regreso a la máquina que lanzó la tarea. Hay diferentes valores por defecto del sistema de transferencia de ficheros dependiendo del Universo y de la plataforma:

- Para las tareas del Universo Standard la existencia de un sistema de ficheros distribuido no es relevante. El acceso a los ficheros (tanto de entrada como de salida) se realiza a través del mecanismo de llamada a sistema remoto (RPC).
- Para las tareas del Universo Vanilla, java, MPI, y Universo paralelo, el acceso a los ficheros, a través de un sistema de archivos distribuido se presume por defecto en las máquinas UNIX. Si no hay ningún sistema de ficheros, el mecanismo de transferencia de ficheros debe ser especificado de manera específica. Cuando se lanza una tarea desde una máquina Windows, Condor asume lo contrario: no hay acceso a un sistema de ficheros distribuido y activa el mecanismo de transferencia por defecto.
- Para el Universo Grid las tareas son ejecutadas en máquinas remotas, por lo que nunca hay un sistema de ficheros compartido entre las máquinas.
- Para el Universo PVM los ficheros que se transfieren no son otros que el ejecutable del **master** y los ficheros dados como comandos de entrada, salida y error no se soportan.
- Para el Universo Scheluder Condor se usa en la máquina desde la que se lanza la tarea.

Para activar el mecanismo de transferencia de ficheros, dos comandos deben de situarse en el fichero de descripción submit:

- **Should_transfer_files**: especifica cuando Condor debe transferir ficheros de entrada desde la máquina que lanzó la tarea, a la máquina remota que la ejecuta. También especifica cuando los ficheros de salida son enviados de regreso a la máquina que lanzó la tarea. El comando puede tener tres valores posibles:
  - YES: Condor siempre transfiere los ficheros de entrada y salida.
  - IF_NEEDED: Condor transfiere ficheros si a la tarea se le asigna una máquina con un FileSystemDomain distinto a la máquina que lanzó la tarea.
  - NO: Deshabilita el sistema de transferencia de ficheros.
- **When_to_transfer_output**: este comando indica cuando deben de mandarse de regreso los ficheros de salida después de que la tarea sea ejecutada en la máquina remota. Sus posibles valores son:
  - ON_EXIT: Condor envía los ficheros de salida a la máquina que lanzó la tarea, cuando la tarea acaba por sí misma.
  - ON_EXIT_OR_EVICT: Condor siempre realizará la transferencia, si la tarea se completa por sí misma, si la máquina recibe una tarea con mayor prioridad, abandona la máquina, o es eliminada. Cuando las tareas se completan, la salida se manda de regreso a la máquina que lanzó la tarea. En los otros casos, los ficheros son transmitidos de regreso a la vez que abandonan la máquina. Estos ficheros son enviados por el directorio dado por la variable SPOOL del fichero de configuración.
- **Transfer_input_files**: si el mecanismo de transmisión de ficheros está activado, Condor enviará los siguientes ficheros, antes de que la tarea sea ejecutada en la máquina remota: el ejecutable, la entrada definida por el comando input, y los ficheros jar en el caso de que se emplee el Universo Java. Si la tarea requiere otro tipo de entrada Condor deberá emplear este comando. Estos ficheros son empleados en el mismo directorio de trabajo temporal, que el ejecutable de la tarea. Para restringir los ficheros que son enviados de regreso, habrá que especificar una lista exacta con el comando **transfer_output_files**.

4.1.4 Variables de entorno

El entorno en el que se ejecuta una tarea normalmente contiene información que es potencialmente útil para la tarea. Condor permite a un usuario referenciar y establecer las variables de entorno para una tarea o un cluster de tareas. A través de un fichero de descripción submit, el usuario debe definir las variables de entrono para el entorno de la tarea usando el comando **enviroment**.

El entorno de la máquina que lanzó la tarea se puede copiar dentro de la ClassAd de la tarea.

4.2 Tipos de tareas

4.2.1 Tareas Dagman

Un grafo acíclico directo (DAG) se puede usar para representar un conjunto de cálculos, donde la entrada, la salida, o la ejecución de una o más tareas es dependiente de una u otra tarea. Las tareas son nodos (vértices) en el grafo, y los arcos identifican las dependencias. Condor encuentra máquinas para la ejecución de los programas, pero no establece la planificación de dichos programas basados en dependencias. DAGMAN es un controlador para la ejecución de programas. DAGMAN lanza las tareas en Condor en el orden representado por el grafo, y procesa el resultado. Un fichero de entrada DAG describe el grafo, y además los ficheros de descripción submit son usados por DAGMAN cuando lanzan tareas para ser ejecutadas por Condor.

DAGMAN es asimismo ejecutado como una tarea del Universo Scheluder a través de Condor. Del mismo modo que DAGMAN lanza tareas, controla los ficheros log para hacer cumplir el orden requerido por el DAG. DAGMAN también es responsable de catalogar, recuperar e informar al conjunto de tareas lanzadas en Condor. Un nodo en un DAG debe abarcar más de un programa lanzado para su ejecución sobre Condor. Una limitación que posee es que todas las tareas del mismo clúster deben usar el mismo fichero log.

Una tarea DAGMAN organiza y lanza las tareas a través de los nodos a Condor, las cuales son definidas para tener éxito o fallar dependiendo de sus valores de retorno. Dicho éxito o fallo se propaga de manera bien definida a nivel del nodo a través de DAG. El fallo de una sola tarea a través de un clúster con múltiples tareas causan que el conjunto entero de las tareas fallen. Todas las tareas del clúster son eliminadas de manera inmediata. Cualquier nodo del DAG está definido para tener éxito o fallar, basado en los valores devueltos por el script PRE, las tareas del clúster, y/o los script POST.

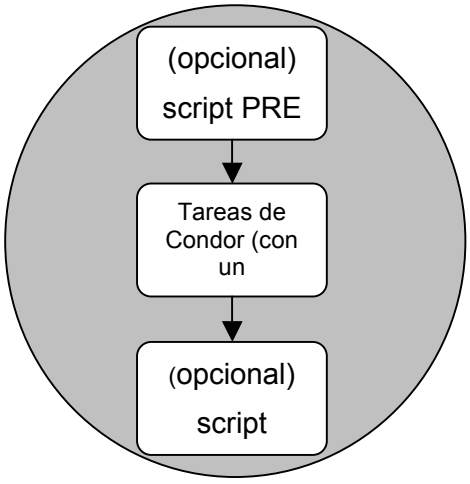


Figura 4. 1 Nodo de una tarea DAGMAN

4.2.2 Tareas Matlab

Matlab es al mismo tiempo un entorno y un lenguaje de programación. Uno de sus puntos fuertes es el hecho de que el lenguaje de Matlab permite construir nuestras propias herramientas reusables. Podemos fácilmente crear nuestras propias funciones y programas especiales (conocidos como archivos .m) en código Matlab. Los podemos

agrupar en Toolbox: colección especializada de archivos .m para trabajar en clases particulares de problemas.

La manera más fácil de visualizar Matlab es pensar en él como en una calculadora totalmente equipada, aunque en realidad, ofrece muchas más características y es mucho más versátil que cualquier calculadora. Matlab es una herramienta para hacer cálculos matemáticos. Es una plataforma de desarrollo de aplicaciones, donde conjuntos de herramientas inteligentes para resolución de problemas en áreas de aplicación específica, a menudo llamadas toolboxes, se pueden desarrollar con una facilidad relativa.

4.2.3 Tareas en C y Fortran

- **Tareas en C**

C es un lenguaje de programación relativamente minimalista. Uno de los objetivos de diseño de este lenguaje fue que sólo fueran necesarias unas pocas instrucciones en lenguaje máquina para traducir cada elemento del lenguaje, sin que hiciera falta un soporte intenso en tiempo de ejecución. Es muy posible escribir C a bajo nivel de abstracción; de hecho, C se usó como intermediario entre diferentes lenguajes. En parte a causa de ser de relativamente bajo nivel y de tener un modesto conjunto de características, se pueden desarrollar compiladores de C fácilmente. En consecuencia, el lenguaje C está disponible en un amplio abanico de plataformas (seguramente más que cualquier otro lenguaje). Además, a pesar de su naturaleza de bajo nivel, el lenguaje se desarrolló para incentivar la programación independiente de la máquina. Un programa escrito cumpliendo los estándares e intentando que sea portable puede compilarse en muchos computadores.

C se desarrolló originalmente (conjuntamente con el sistema operativo Unix, con el que ha estado asociado mucho tiempo) por programadores para programadores. Sin embargo, ha alcanzado una popularidad enorme, y se ha usado en contextos muy alejados de la programación de sistemas, para la que se diseñó originalmente.

- **Tareas Fortran**

El lenguaje fue diseñado tomando en cuenta que los programas serían escritos en tarjetas perforadas de 80 columnas. Así por ejemplo, las líneas debían ser numeradas y la única alteración posible en el orden de ejecución era producida con la instrucción goto. Estas características han evolucionado de versión en versión. Las versiones actuales contienen subprogramas, recursión y una variada gama de estructuras de control.

Lo que fue la primera tentativa de proyección de un lenguaje de programación de alto nivel, tiene una sintaxis considerada arcaica por muchos programadores que aprenden lenguajes más modernos. Es difícil escribir un bucle "for", y errores en la escritura de sólo un carácter pueden llevar a errores durante el tiempo de ejecución en vez de errores de compilación, en el caso de que no se usen las construcciones más frecuentes. Algunas de las versiones anteriores no poseían facilidades que son consideradas como útiles en las máquinas modernas, como la colocación dinámica de memoria. Se debe tener en cuenta que la sintaxis de Fortran fue afinada para el uso en trabajos numéricos y científicos y que muchas de sus deficiencias han sido abordadas en revisiones más recientes del lenguaje. Por ejemplo, Fortran 95 posee comandos mucho más breves para efectuar operaciones matemáticas con matrices y dispone de tipos. Esto no sólo mejora mucho la lectura del programa sino que además aporta información útil al compilador. Por estas razones Fortran no es muy usado fuera de los campos de la informática y el análisis numérico, pero permanece como el lenguaje a escoger para desempeñar tareas de computación numérica de alto rendimiento.

4.3 Proceso de lanzamiento de tareas.

4.3.1 Tareas Dagman

4.3.1.1 Fichero de entrada que describe el DAG

Describe siete tipos de ítems:

1. Una lista de nodos en el DAG que causa el lanzamiento de una o más tareas de Condor. Cada entrada sirve para nombrar un nodo y se refiere a un fichero de descripción submit.
2. Una lista de nodos en el DAG que causa el envío de datos de tareas en cola. Cada entrada se emplea para nombrar un nodo y especificar el fichero de descripción submit Stork.
3. Cualquier proceso que necesita situarse en un nodo de condor o una tarea Stork.
4. Una descripción de las dependencias del DAG.
5. El número de veces que se reintenta la ejecución de un nodo, si un nodo del DAG falla.
6. Cualquier definición de macro asociada a un nodo.
7. Un valor del nodo de salida que causa que todo el DAG falle.

Pueden incluirse comentarios en el fichero de entrada del DAG, poniendo de antemano el carácter #.

Comandos del fichero de entrada:

JOB

El primer conjunto de líneas en el fichero de entrada DAG describe a cada una de las tareas que aparecen en el DAG. Cada nodo que va a ser manejado por Condor se describe como una sola línea que comienza con la palabra clave JOB. La sintaxis empleada para cada entrada JOB es:

JOB nombre_de_la_tarea Nombre_del_fichero_de_descripcion_submit[done]

Una entrada JOB describe el nombre de la tarea del fichero de descripción submit. El nombre de la tarea sólo identifica nodos a través de los ficheros de entrada de DAGMAN y los mensajes de salida.

El parámetro DONE identifica una tarea que ha sido completada. Es útil en situaciones donde los usuarios desean verificar resultados, sin la necesidad de ejecutar todas las tareas del grafo. La macro DONE también se emplea cuando ocurre un error que hace que el grafo finalice sin completarse. DAGMAN genera un DAG Rescue, un fichero de entrada DAG que puede usarse para reiniciar y completar un DAG sin reejecutar los nodos completos, sólo la parte que falló.

DATA

Identifica una tarea que debe ser manejada por el servidor de datos Stork. La sintaxis empleada para cada entada DATA es:

DATA JobName SubmitDescriptionFileName [DONE]

Una entrada DATA traza el mapa del nombre de una tarea con un fichero de descripción submit Store. En el resto de los aspectos el comando DATA es idéntico al del comando JOB.

SCRIPT

Identifica los procesamientos realizados antes de que una tarea del DAG se lance a Condor o a Stork para su ejecución, o después de que una tarea del DAG complete su ejecución. Los procesos que se realicen antes de que se ejecute una tarea se llama script PRE. Los procesos que se producen después de que las tareas completen su ejecución se llaman script POST.

SCRIPT PRE JobName ExecutableName[arguments]

SCRIPT POST JobName ExecutableName[arguments]

Las palabras PRE o POST especifican el cronometraje de cuando debe ejecutarse el script. El parámetro JobName especifica el nodo en el que se ejecuta el script. El ExecutableName especifica el script que va a ser ejecutado.

PARENT..CHILD

Especifican las dependencias del DAG. Los nodos son padres o hijos en el DAG. Un nodo padre debe completarse de manera satisfactoria, antes de que uno de sus hijos comience. Un nodo hijo sólo empezará una vez que todos sus padres hayan finalizado. La sintaxis es:

PARENT ParentJobName…CHILD ChildJobName…

RETRY

Es una manera de reintentar los nodos fallidos

RETRY JobName NumberOfRetries[UNLESS_EXIT<value>]

Donde JobName identifica el nodo y NumberOfRetries es un entero, que indica el número de reintentos del nodo después del fallo

VARS

Se usa para definir macros. Su sintaxis es la que sigue:

VARS JobName macroname=”string”[macroname=”string”…]

Esta macro puede ser usada en un fichero de descripción submit

ABORT-DAG-ON

Elimina el DAG por completo si un nodo dado, devuelve un valor específico. Su sintaxis es:

ABORT-DAG-ON JobName AbortExitValue [return DAGReturnValue]

Si el nodo especificado por el parámetro JobName retorna el valor específico AbortExitValue, el DAG aborta su ejecución. Un DAG que se aborta, a diferencia de un nodo fallido, hace que todos los nodos del DAG paren su ejecución inmediatamente. Esto incluye eliminar las tareas de los nodos que están ejecutándose. Un nodo fallido permite que DAGMAN continúe ejecutándose, aunque no podrán realizarse más procesos debido a las dependencias. Este comando también elimina las dependencias entre los nodos. El DAG para, aunque el nodo posea reintentos.

4.3.1.2 Fichero de descripción Submit

Cada nodo del DAG usa un único fichero de descripción submit. Una limitación clave, es que cada fichero de descripción submit de Condor debe lanzar tareas descritas por un único número de clúster.

Todas las tareas de todos los nodos generan ficheros de descripción log. Sin embargo el rendimiento puede verse afectado, por lo que es mejor reducir el número de ficheros log.

4.3.1.3 Tarea Propuesta

El fichero de descripción submit de la tarea propuesta es el siguiente:

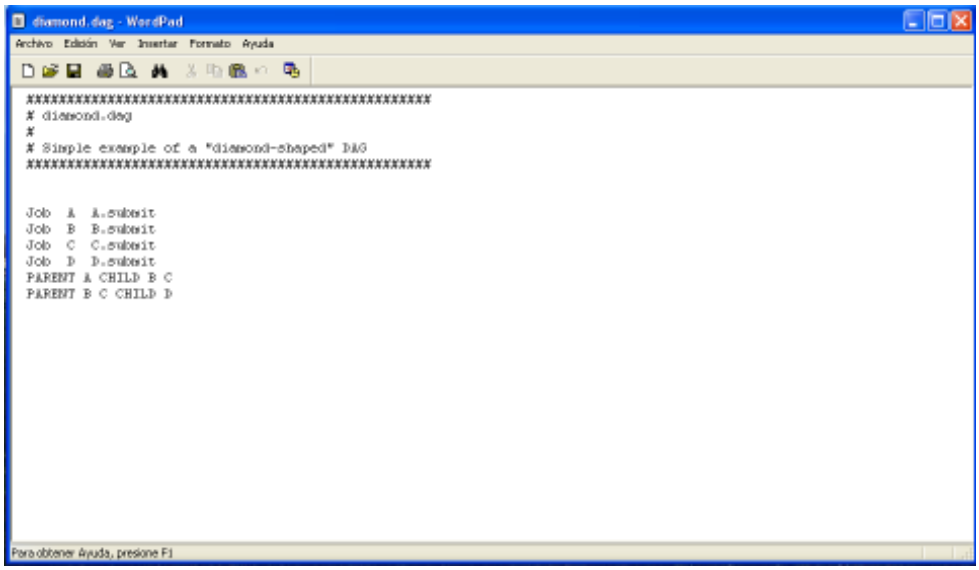
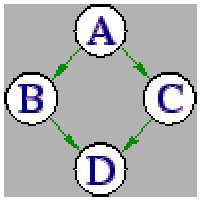


Figura 4. 2 Fichero de descripción submit de Dagman

Dicho fichero indica la siguiente relación:



Este DAG posee cuatro tareas: A. B, C, D. Las tareas poseen las siguientes dependencias: A es el padre, B y C son hijos de A, y D es hijo de B y C.

Los ficheros de descripción submit de las tareas independientes son los siguientes:

TAREA A

```
####################################################
# Fichero submit para la tarea A
####################################################

Universe   = vanilla
Executable = random.condor
output     = A.out
log        = diamond.log
Queue
```

En esta tarea, el ejecutable random.condor se refiere a un programa en C que genera un número aleatorio divisible por 2, y escribe el resultado en el archivo A.out,

descrito en la entrada output. Este programa se ejecuta en el universo Vanilla. Las características de la ejecución se muestran en el archivo log diamond.log, y la tarea se pone en cola gracias al comando Queue.

TAREA B

```
##################################################
# Fichero submit para la tarea B
#
# half.condor lee un número desde STDIN,
# lo divide por 2, y muestra el resultado en STDOUT.
##################################################

Universe   = vanilla
Executable = half.condor
input      = A.out
output     = B.out
log        = diamond.log
Queue
```

En esta tarea el ejecutable half.condor, toma una entrada desde el archivo A.out, cuyo contenido es un entero, divide dicho entero por 2, y escribe el resultado en el fichero B.out. Al igual que la tarea anterior, sus características se muestran en el archivo diamond.log, y la tarea se pone en cola con el comando Queue.

TAREA C

```
##################################################
# Fichero submit para la tarea C
##################################################

Universe   = vanilla
Executable = half.condor
input      = A.out
output     = C.out
log        = diamond.log
Queue
```

Al igual que la tarea B, toma el entero procedente del archivo A.out, lo divide entre 2 mediante el programa en C half.condor, y escribe el resultado en el archivo C.out, por lo que posee la misma salida que B.out. También muestra el proceso de ejecución en el archivo diamond.log y pone la tarea en cola con el comando Queue.

TAREA D

```
##################################################
# Submit file for Job D
##################################################

Universe   =vanilla
Executable =sum.condor
arguments  = B.out C.out
output     = D.out
error      = D.err
log        = diamond.log
Queue
```

La tarea D ejecuta un programa en C denominado sum.c que toma los archivos procedentes de B.out y C.out, los suma y escribe el resultado en la salida D.out, que es el mismo número que el que se creó en el archivo A.out. Si se produce algún error, éste se mostrará en el archivo D.err. Para finalizar, las características de la ejecución se muestran también en el archivo diamond.log y la tarea se pone en cola mediante el comando Queue.

4.3.1.4 Lanzamiento de la tarea

Lanzamiento de la tarea en Windows

Para lanzar la tarea el primer paso es la compilación de los archivos. En Windows no existe un ejecutable condor_compile como en el caso de Redhat. Para realizar la compilación de los archivos en C se ejecutará lo siguiente:

gcc –o random.condor random.c

gcc –o half.condor half.c

gcc –o sum.condor sum.c

Esto compilará los archivos y generará los ejecutables random.condor, half.condor y sum.condor necesarios para poder realizar la tarea descrita. Para poder realizar la tarea en Condor hay que ejecutar:

condor_submit_dag diamond.dag

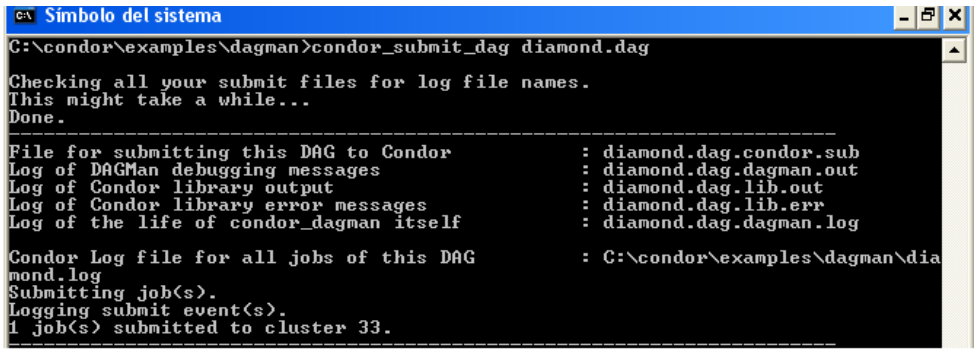


Figura 4. 3 Lanzamiento de la tarea Dagman

Lanzamiento de las tarea en Redhat

La principal diferencia entre el lanzamiento de la tarea en un entorno u otro está en la compilación de los archivos. En Redhat se posee el binario condor_compile que enlaza las librerías de Condor con las tareas a lanzar.

El archivo Makefile para la compilación de las tareas del grafo sería el siguiente:

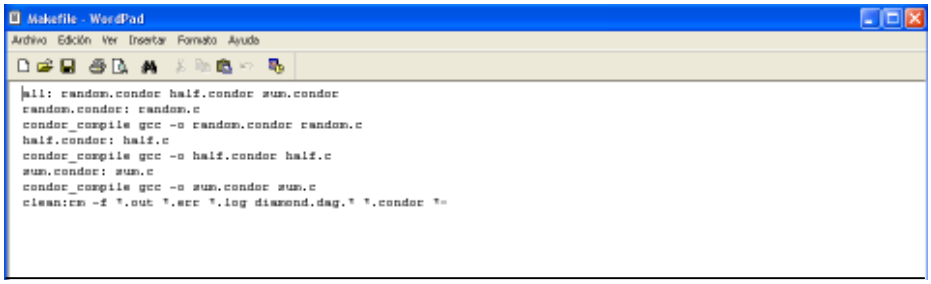


Figura 4. 4 Archivo Makefile de la tarea Dagman

Por lo que para lanzar la tarea se deberá ejecutar lo siguiente:

[condor@maquina2 dagman]$ make

[condor@maquina2 dagman]$ condor_submit_dag diamond.dag

El resultado de la ejecución de estos comandos es el mismo que pudo observarse en la Figura 4.3. Los archivos que se crean se muestran en la figura siguiente:

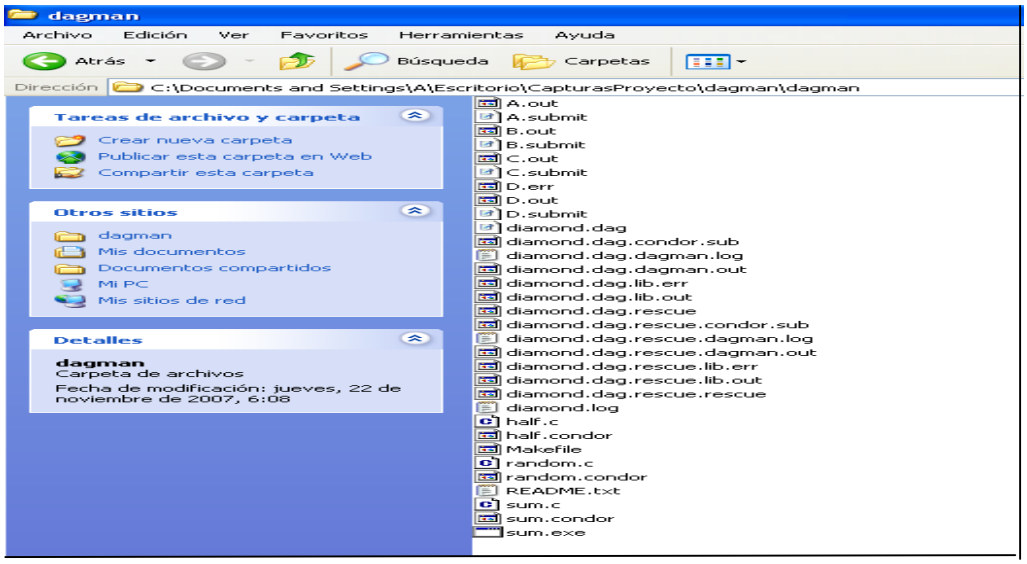


Figura 4. 5 Ficheros creados tras la compilación y ejecución de los archivos Dagman

4.3.2 Tareas Matlab

Antes de comenzar con la explicación sobre el lanzamiento y ejecución de las tareas, es necesario indicar que durante la realización de este proyecto no pudo demostrarse el funcionamiento de este tipo de tareas en Condor.

Tras mucho investigar en libros y foros de Internet, se llegó a la conclusión de que el problema era dependiente de las propias máquinas. Aún así a continuación se explica como se realizaría la creación y lanzamiento de una tarea en Matlab.

4.3.2.1 Tarea propuesta

El primer paso es la creación de un fichero de descripción de la tarea (fichero .m), que contenga la propia tarea a ejecutar. En el caso ejemplo dicha tarea consiste en el cálculo del factorial del número 13, llamándose el archivo *fact.m* que es el que se describe a continuación:

```
factorial(13)
quit
```

El siguiente paso es la creación del archivo ejecutable:

Para el caso de RedHat (matlab.sh):

```
#!/bin/bash
/usr/local/bin/matlab -nojvm -nodisplay -nosplash -r fact.m
```

Para el caso de Windows (matlab.bat):

```
set path=%path%;c:\Archivos de programa\MATLAB\R2007a\bin\win32
MATLAB.exe -nojvm -nodisplay -nosplash -r fact.m
```

Para el caso de Redhat, la ejecución de la tarea se produce de manera directa, pero en el caso de Windows (archivo matlab.bat) es necesario especificar el path de Matlab.

Una vez hecho esto se procede a la creación del fichero de descripción submit llamado *matlab.sub*, que será el que propiamente se lance en Condor. Uno de los puntos calves de estos ficheros es que el único universo posible en el que la tarea podrá lanzarse es el universo Vanilla, ya que Matlab sólo admite este universo para su ejecución:

Para el caso de RedHat:

```
Universe= Vanilla
Requierements= HAS_MATLAB =?= TRUE && OpSys == "LINUX" && Arch ==
"x86"
Enviroment= HOME=/home/condor
Executable=matlab.sh
Arguments=fact
Should_transfer_files=YES
When_to_transfer_output=ON_EXIT
Input=fact.m
Output=Matlab.out
Log=Matlab.log
Error=Matlab.err
Queue
```

Para el caso de Windows:

```
Requierements= HAS_MATLAB =?= TRUE && OpSys == "WINNT51" && Arch ==
"x86"
Enviroment= c:\condor\bin\examples
Executable=matlab.bat
Arguments=fact
Should_transfer_files=YES
When_to_transfer_output=ON_EXIT
Input=fact.m
Output=Matlab.out
Log=Matlab.log
Error=Matlab.err
Queue
```

En ambos archivos se toma como argumento de entrada *Input* el fichero *fact.m*, escribiendo el resultado de la salida *Output* en el fichero *Matlab.out*, y generando los archivos de salida *Matlab.log*, que posee un resumen de los sucesos de la ejecución, y *Matlab.err* que mostrará los fallos en el caso de que hubieran.

Para poder aceptar los ficheros de entrada, es necesario habilitar *Should_transfer_files*, y para poder obtener la salida desde Matlab, debe habilitarse *When_to_transfer_output*, que en este caso al establecerse como *ON_EXIT*, sólo mostrará el resultado tras la finalización completa de la tarea. En *arguments* se encuentra el archivo compilado de fact.m.

La ejecución de tareas Matlab en RedHat y Windows se diferencian en que en este último el ejecutable *Executable* es *matlab.bat,* y para Redhat es *matlab.sh*. Además en la especificación de las características, *Requierements*, que se desean que las máquinas de ejecución posean, en el caso de Redhat hay que poner como sistema operativo *OpSys LINUX*, y en el caso de Windows hay que poner *WINNT51*.

4.3.2.2 Lanzamiento de la tarea

Para lanzar la tarea, los pasos son los mismos en Linux y en Windows: El primer paso es la compilación de la misma que se realiza mediante el siguiente comando:

```
[condor@maquina2 examples]$ mcc -mv fact.m
```

Para lanzar la tarea habrá que ejecutar lo siguiente:

```
[condor@maquina2 examples]$ condor_submit matlab.sub
```

Una vez hecho esto Condor se encargará de gestionar la ejecución de la tarea, y se crearán los archivos explicados anteriormente

4.3.3 Tareas C y Fortran

Las tareas C y Fortran, son las más sencillas de implementar. Sólo se diferencian en el proceso de lanzamiento de la tarea, en el comando que se usa para compilar. Por ello se va a describir el lanzamiento de una tarea en C, indicándose en que puntos de diferencia de Fortran, y en que puntos se diferencian los sistemas operativos.

4.3.3.1 Tarea Propuesta

El primer paso para el lanzamiento de una tarea en C o en Fortran es la creación de la propia tarea. En el caso de ejemplo de C, la tarea loop.c consiste en un bucle que imprime cinco archivos (lanza 5 tareas), que imprimen números del 1 al 200 en dos de los archivos, de uno a 300 en otros dos y de 1 a 500 en el último.

El archivo de descripción submit loop.submit de la tarea es el siguiente:

```
universe=vanilla
requeriments=(Arch=="Intel")&&(OpSys=="WINNT51")
environment=path=c:\Windows\system32
executable  = c:\condor\examples\loop.exe
output            = loop.$(Process).out
error       = loop.$(Process).err
log         = loop.log
arguments   = 200
queue 2
arguments   = 300
queue 2
arguments   = 500
output            = loop.last.out
error       = loop.last.err
queue
```

Para poder lanzar las tareas en Linux habría que cambiar:

```
requeriments=(Arch=="Intel")&&(OpSys=="LINUX")
environment=/home/condor
executable=loop.remote
```

Los parámetros que se muestran son los mismos que se explicaron en las tareas de matlab. Queue 2 hace que 2 tareas del mismo ejecutable se pongan en cola.

4.3.3.2 Lanzamiento de la tarea

Para lanzar la tarea, los pasos son los mismos en Linux y en Windows salvo a la hora de compilar:

Para el caso de Linux:

[condor@maquina2 examples]$ `condor_compile gcc -o loop.remote loop.c`

Si la tarea fuera de Fortran la única diferencia sería a la hora de compilar:

[condor@maquina2 examples]$ `condor_compile g77 -o tarea.remote tarea.f`

Para el caso de Windows:

C:\condor\examples> gcc –o loop.exe loop.c

Para lanzar la tarea habrá que ejecutar lo siguiente tanto en Linux como en Windows:

[condor@maquina2 examples]$ `condor_submit loop.sub`

Una vez hecho esto Condor se encargará de gestionar la ejecución de la tarea, y se crearán los archivos de salida correspondientes.

4.4 Proceso de ejecución de tareas Dagman.

Para poder visualizar la tarea que se lanzó mediante línea de comandos los comandos más útiles son condor_status que nos mostrará el estado de las máquinas que forman parte del Pool, y condor_q que nos mostrará el proceso de ejecución de las mismas:

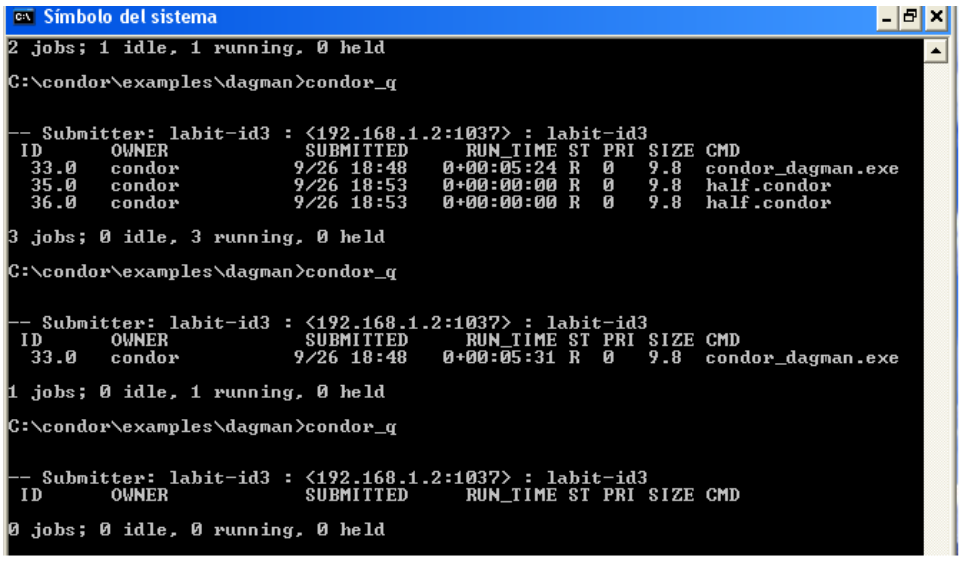


Figura 4. 6 Visualización del estado de las tareas mediante condor_q

Otra manera de poder ver el progreso de las tareas es mediante el archivo diamond.dag.dagman.out, que muestra paso por paso el progreso de una tarea, indicando, si ocurre, cuando se produce un fallo. Un ejemplo de parte de este archivo es el siguiente:

```
9/26 18:48:26 ******************************************************
9/26 18:48:26 ** condor_scheduniv_exec.33.0 (CONDOR_DAGMAN) STARTING
UP
9/26 18:48:26 ** c:\condor\bin\condor_dagman.exe
9/26 18:48:26 ** $CondorVersion: 6.8.4 Feb  1 2007 $
9/26 18:48:26 ** $CondorPlatform: INTEL-WINNT50 $
9/26 18:48:26 ** PID = 1644
9/26 18:48:26 ** Log last touched time unavailable (No such file or
directory)
9/26 18:48:26 ******************************************************
9/26 18:48:26 Using config source: C:\condor\condor_config
9/26 18:48:26 Using local config sources:
9/26 18:48:26    C:\condor/condor_config.local
9/26 18:48:26 DaemonCore: Command Socket at <192.168.1.2:1248>
9/26 18:48:26 DAGMAN_SUBMIT_DELAY setting: 0
9/26 18:48:26 DAGMAN_MAX_SUBMIT_ATTEMPTS setting: 6
9/26 18:48:26 DAGMAN_STARTUP_CYCLE_DETECT setting: 0
9/26 18:48:26 DAGMAN_MAX_SUBMITS_PER_INTERVAL setting: 5
9/26 18:48:26 allow_events (DAGMAN_IGNORE_DUPLICATE_JOB_EXECUTION,
DAGMAN_ALLOW_EVENTS) setting: 114
9/26 18:48:26 DAGMAN_RETRY_SUBMIT_FIRST setting: 1
9/26 18:48:26 DAGMAN_RETRY_NODE_FIRST setting: 0
9/26 18:48:26 DAGMAN_MAX_JOBS_IDLE setting: 0
9/26 18:48:26 DAGMAN_MAX_JOBS_SUBMITTED setting: 0
9/26 18:48:26 DAGMAN_MUNGE_NODE_NAMES setting: 1
9/26 18:48:26 DAGMAN_DELETE_OLD_LOGS setting: 1
9/26 18:48:26 DAGMAN_PROHIBIT_MULTI_JOBS setting: 0
9/26 18:48:26 DAGMAN_ABORT_DUPLICATES setting: 0
9/26 18:48:26 argv[0] == "condor_scheduniv_exec.33.0"
9/26 18:48:26 argv[1] == "-Debug"
9/26 18:48:26 argv[2] == "3"
9/26 18:48:26 argv[3] == "-Lockfile"
9/26 18:48:26 argv[4] == "diamond.dag.lock"
9/26 18:48:26 argv[5] == "-Condorlog"
9/26 18:48:26 argv[6] == "C:\condor\examples\dagman\diamond.log"
9/26 18:48:26 argv[7] == "-Dag"
9/26 18:48:26 argv[8] == "diamond.dag"
9/26 18:48:26 argv[9] == "-Rescue"
9/26 18:48:26 argv[10] == "diamond.dag.rescue"
9/26 18:48:26 DAG Lockfile will be written to diamond.dag.lock
9/26 18:48:26 DAG Input file is diamond.dag
9/26 18:48:26 Rescue DAG will be written to diamond.dag.rescue
9/26 18:48:26 All DAG node user log files:
9/26 18:48:26   C:\condor\examples\dagman\diamond.log (Condor)
9/26 18:48:26 Parsing diamond.dag ...
9/26 18:48:26 Dag contains 4 total jobs
9/26 18:48:26 Deleting any older versions of log files...
9/26 18:48:26 Bootstrapping...
9/26 18:48:26 Number of pre-completed nodes: 0
9/26 18:48:26 Registering condor_event_timer...
9/26 18:48:27 Got node A from the ready queue
9/26 18:48:27 Submitting Condor Node A job(s)...
9/26 18:48:27 submitting: condor_submit -a dag_node_name' '=' 'A -a
+DAGManJobId' '=' '33 -a DAGManJobId' '=' '33 -a submit_event_notes'
'=' 'DAG' 'Node:' 'A -a +DAGParentNodeNames' '=' '"" A.submit
9/26 18:48:28 From submit: Submitting job(s). 9/26 18:48:28 From
submit: Logging submit event(s). 9/26 18:48:28 From submit: 1 job(s)
submitted to cluster 34. 9/26 18:48:28   assigned Condor ID (34.0)
9/26 18:48:28 Just submitted 1 job this cycle...
9/26 18:48:28 Event: ULOG_SUBMIT for Condor Node A (34.0)
```

```
9/26 18:48:28 Number of idle job procs: 1
9/26 18:48:28 Of 4 nodes total:
9/26 18:48:28  Done     Pre   Queued    Post   Ready   Un-Ready
Failed
9/26 18:48:28   ===      ===      ===      ===      ===         ===
===
9/26 18:48:28     0        0        1        0        0           3
0
9/26 18:53:33 Event: ULOG_EXECUTE for Condor Node A (34.0)
9/26 18:53:33 Number of idle job procs: 0
9/26 18:53:33 Event: ULOG_JOB_TERMINATED for Condor Node A (34.0)
9/26 18:53:33 Node A job proc (34.0) completed successfully.
9/26 18:53:33 Node A job completed
9/26 18:53:33 Number of idle job procs: 0
9/26 18:53:33 Of 4 nodes total:
9/26 18:53:33  Done     Pre   Queued    Post   Ready   Un-Ready
Failed
9/26 18:53:33   ===      ===      ===      ===      ===         ===
===
9/26 18:53:33     1        0        0        0        2           1
0
```

En este caso se puede observar como se ponen en la cola las cuatro tareas, y empieza con el lanzamiento de la primera tarea, en este caso la tarea A, que se añade al clúster 34, mostrándose que uno de los nodos está en la cola (Queued) y hay tres tareas hijas de la tarea A que no están listas para ser usadas (Un-Ready). Una vez que la tarea A se completa, las tareas B y C pasan a estas listas para ser completadas (Ready), quedando únicamente a la espera la tarea D (Un-Ready). En el array argv se muestra la localización de los archivos necesarios para llevar a cabo la ejecución de las tareas. Una manera más sencilla de comprobar el éxito o fracaso de una tarea es mediante los archivos .*err*, que a menos que ocurra algo, tendrán un tamaño nulo.

4.5 Obtención de resultados.

4.5.1 Tareas dagman

Cuando se termina la ejecución de la tarea pueden ocurrir dos cosas:

- Si la tarea termina con éxito, se obtendrá un archivo de salida por cada tarea que se desee, si así se ha especificado, o bien un único archivo de salida con el resultado final. En el caso de la tarea que hemos planteado se obtiene un archivo por cada tarea que se ha planteado: A.out, B.out, C.out, D.out.

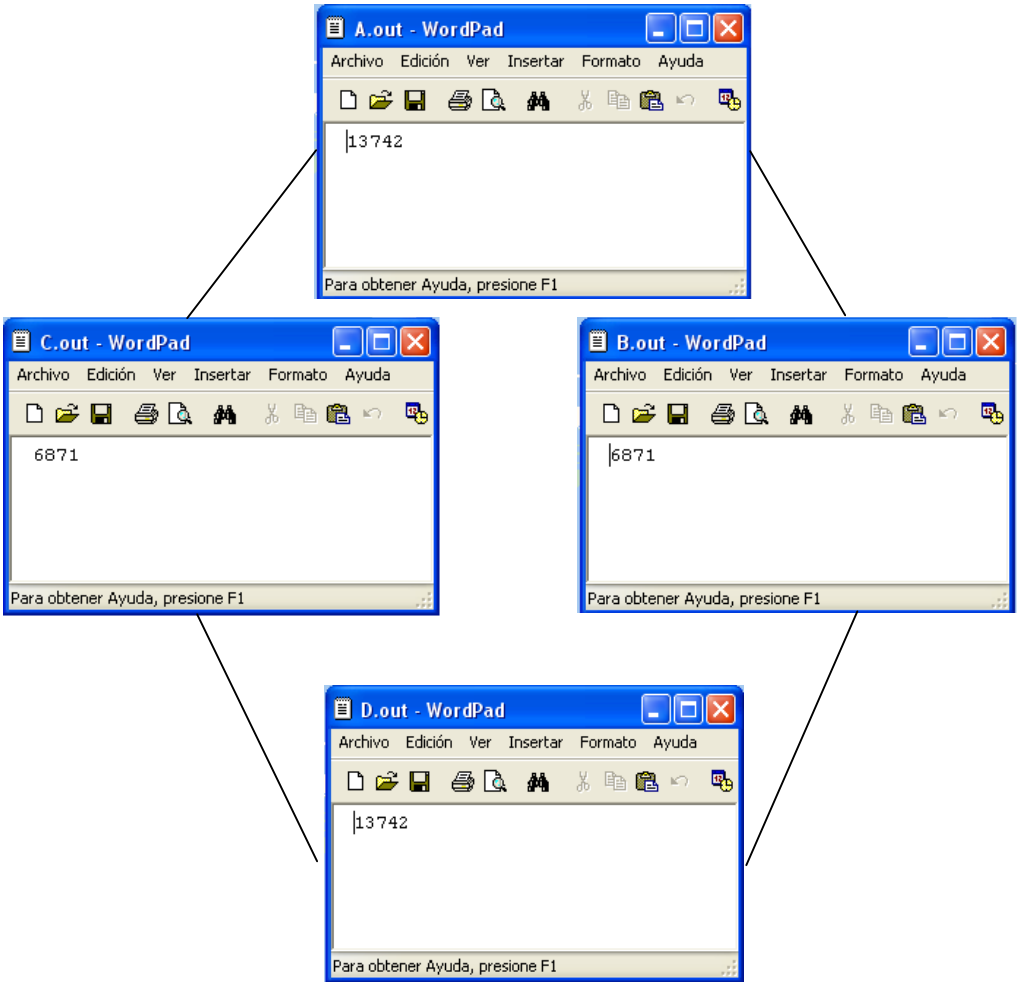


Figura 4. 7 Archivos de salida de las tareas Dagman

- Si hay algún problema el archivo dagman.out mostrará una salida parecida a la siguiente:

```
9/26 18:54:03 Got node D from the ready queue
9/26 18:54:03 Submitting Condor Node D job(s)...
9/26 18:54:03 submitting: condor_submit -a dag_node_name' '=' 'D -a
+DAGManJobId' '=' '33 -a DAGManJobId' '=' '33 -a submit_event_notes' '='
'DAG' 'Node:' 'D -a +DAGParentNodeNames' '=' '"B,C" D.submit
9/26 18:54:04 From submit: Submitting job(s). 9/26 18:54:04 From submit:
Logging submit event(s). 9/26 18:54:04 From submit: 1 job(s) submitted to
cluster 37. 9/26 18:54:04     assigned Condor ID (37.0)
9/26 18:54:04 Just submitted 1 job this cycle...
9/26 18:54:04 Event: ULOG_SUBMIT for Condor Node D (37.0)
9/26 18:54:04 Number of idle job procs: 1
9/26 18:54:04 Of 4 nodes total:
9/26 18:54:04  Done     Pre   Queued    Post   Ready   Un-Ready   Failed
9/26 18:54:04   ===     ===      ===     ===     ===         ===      ===
9/26 18:54:04     3       0        1       0       0           0        0
9/26 18:54:14 Event: ULOG_EXECUTE for Condor Node D (37.0)
9/26 18:54:14 Number of idle job procs: 0
9/26 18:54:14 Event: ULOG_JOB_TERMINATED for Condor Node D (37.0)
9/26 18:54:14 Node D job proc (37.0) failed with status 1.
9/26 18:54:14 Number of idle job procs: 0
9/26 18:54:14 Of 4 nodes total:
9/26 18:54:14  Done     Pre   Queued    Post   Ready   Un-Ready   Failed
```

```
9/26 18:54:14   ===      ===       ===      ===      ===         ===       ===
9/26 18:54:14     3        0         0        0        0           0         1
9/26 18:54:14 ERROR: the following job(s) failed:
9/26 18:54:14 ---------------------- Job ----------------------
9/26 18:54:14       Node Name: D
9/26 18:54:14          NodeID: 3
9/26 18:54:14     Node Status: STATUS_ERROR
9/26 18:54:14 Node return val: 1
9/26 18:54:14           Error: Job proc (37.0) failed with status 1
9/26 18:54:14 Job Submit File: D.submit
9/26 18:54:14   Condor Job ID: (37)
9/26 18:54:14       Q_PARENTS: 1, 2, <END>
9/26 18:54:14       Q_WAITING: <END>
9/26 18:54:14      Q_CHILDREN: <END>
9/26 18:54:14 ---------------------------------------<END>
9/26 18:54:14 Aborting DAG...
9/26 18:54:14 Writing Rescue DAG to diamond.dag.rescue...
9/26 18:54:14 Note: 0 total job deferrals because of -MaxJobs limit (0)
9/26 18:54:14 Note: 0 total job deferrals because of -MaxIdle limit (0)
9/26 18:54:14 Note: 0 total PRE script deferrals because of -MaxPre limit
(0)
9/26 18:54:14 Note: 0 total POST script deferrals because of -MaxPost
limit (0)
9/26 18:54:14 **** condor_scheduniv_exec.33.0 (condor_DAGMAN) EXITING
WITH STATUS 1
```

En el archivo se indica que se ha producido un error, indicando que la tarea ha finalizado con STATUS 1. Además se crea el archivo *.rescue*, que puede emplearse para intentar solventar los posibles problemas del grafo. En dicho archivo se muestra con el comando DONE aquellas tareas finalizadas con éxito, para que en la próxima ejecución sólo intenten ejecutarse las tareas que fallaron:

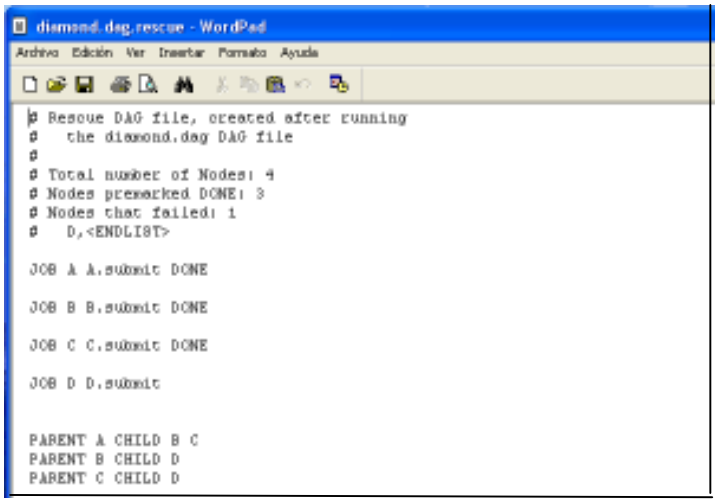


Figura 4. 8 Archivo *.rescue* de Dagman

Una vez arreglado el posible fallo habría que ejecutar lo siguiente:

[condor@maquina2 dagman]$ condor_submit_dag diamond.dag.rescue

De este modo solo intentará ejecutar aquellas tareas que fallaron.

4.5.2 Tareas Matlab

Como ya se explicó en el punto 4.3.2, no ha sido posible comprobar el funcionamiento de Matlab en Condor, pero si se hubiera conseguido, los comandos serían condor_q y condor_status, que fueron los que se usaron en las tareas Dagman, así como el archivo Matlab.log.

4.5.3 Tareas en C y Fortran

Para ver la evolución de la tarea propuesta son útiles los comandos condor_status y condor_q, como ya explicó en las tareas anteriores. Los resultados de los archivos son muy parecidos.

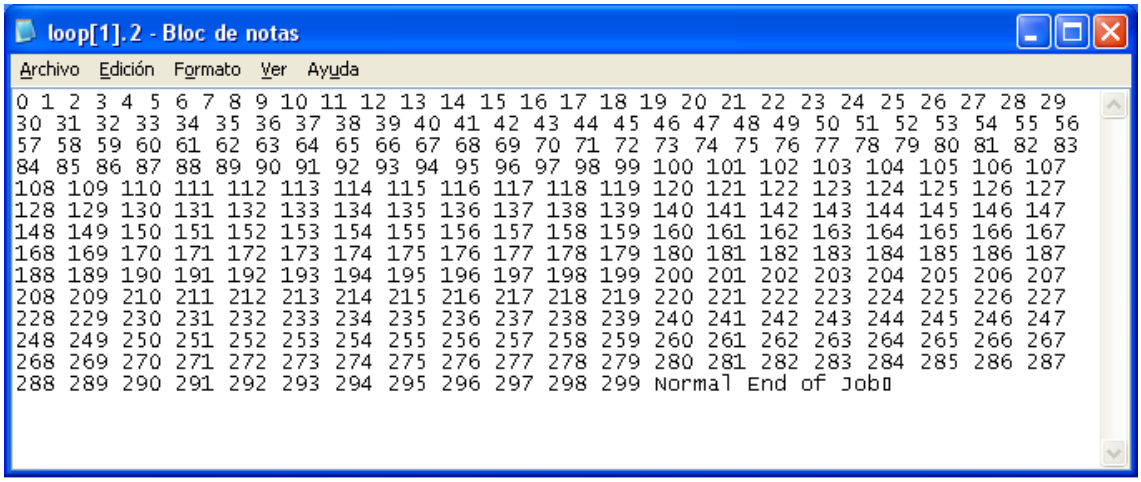


Figura 4. 9 Ejemplo de la salida de un archivo en C

Capítulo 5
Conclusiones y líneas futuras

5.1 Conclusiones

En el primer capítulo de este proyecto se plantearon diferentes objetivos a cumplir en este proyecto final de carrera. Dichos objetivos de idearon a partir de la idea de implementar un entorno de alto rendimiento computacional para aprovechar los recursos computacionales de los que dispone el Área de Ingeniería Telemática sin que ello supusiera un coste extra de hardware o de software y de forma totalmente transparente al usuario, es decir, el personal docente e investigador del área.

En este capitulo se analiza si dichos objetivos han sido cubiertos, extrayendo conclusiones sobre los distintos entornos computacionales existentes, la elección de Condor, el funcionamiento del mismo a nivel usuario y a nivel procesos, sistemas de ficheros, etc, el hardware y el software disponible y utilizado para llevar a cabo el proyecto, los lenguajes de programación usados y los resultados obtenidos tras la implementación del entorno computacional.

En el Capitulo 2 se hizo una valoración de los entornos de computación que hasta el momento existen. Entre los distintos entornos HPC, HTC y GRID se han ido analizando cada uno de ellos hasta llegar a la elección de Condor como entorno HTC que más convenía a las especificaciones de requisitos que se tenían a disposición. En dicho capitulo también se han descrito las características principales de Condor, la arquitectura y modo de funcionamiento así como los distintos “universos” en los que se puede trabajar.

En el Capítulo 3 se describen los pasos que se siguieron para la correcta instalación de un entorno Condor tanto en un S.O. Windows como en Linux así como la puesta en marcha de los procesos (demonios) del entorno. Hay que destacar que para Condor solo se puede instalar en los sistemas operativos indicados en su página. En un intento por trabajar bajo las condiciones de software impuestas en muchos de los laboratorios del área, se intentó instalar Condor-RedHat bajo un entorno Suse, que era el que había instalado en los PCs del laboratorio, lo que desembocó en varios problemas de compatibilidades en archivos de Condor. Además, también surgieron problemas en cuanto a los archivos de configuración de Condor, ya que la aplicación tiene varios archivos de configuración y líneas adicionales que se le pueden añadir para añadir funcionalidades a la aplicación.

En el capitulo 4, y previa comprobación de que el entorno de computación Condor funcionaba, se realizó un estudio sobre que tipo de tares se podían lanzar y cómo debían lanzarse, además de recopilar toda la información referente a que requisitos imponía el lanzamiento, ejecución y recopilación de datos obtenidos de una tarea.

Como se puede observar en los distintos capítulos que conforman este proyecto, se han cumplido satisfactoriamente los objetivos que se marcaron en la sección objetivos del capítulo de Introducción, quedando la aplicación a futuras ampliaciones que se enumeran en la siguiente sección.

5.2 Líneas futuras

Teniendo en cuenta que el proyecto se ha realizado sobre una red prototipo, la línea futura "inmediata" será la migración de lo realizado en la red prototipo a uno de los laboratorios docentes del Área de Ingeniería Telemática.

Tras esa migración, y utilizando la aplicación realizada en un proyecto paralelo a este, el personal docente e investigador podría acceder al Condor Pool creado en el laboratorio via web, sin tener que conocer y/o aprender el funcionamiento, los procesos o los comandos relativos a Condor.

Otra de las líneas futuras inminentes sería, una vez comprobado que en el laboratorio todo funciona correctamente, instalar tantos Condor Pools como laboratorios docentes tiene el Área de Ingeniería Telemática, creando así, por medio del mecanismo Flocking explicado en el apéndice de este proyecto un Condor Grid. De esta forma, a la hora de lanzar las tareas, los usuarios podrían decidir el Condor Pool donde desean lanzarlas, o la arquitectura de las máquinas donde deben ejecutarse esas tareas, etc.

Siguiendo con la expansión de Condor Grid y el aprovechamiento de los recursos desaprovechados de la Escuela de Telecomunicaciones, podría crearse un gran Condor GRID en la escuela, utilizando, no solo los laboratorios del AIT sino también el resto de laboratorios que forman parte de la escuela. Este aumento de la capacidad de procesamiento podría llegar hasta el orden de TFLOPS TFLOPS (10^{12} FLOPS). De esta forma, no solo se aprovecharían al máximo los recursos de la escuela sino que también se diseñaría un GRID con diversidad de Condor Pools, cada uno implementado con un universo distintos, diversidad de sistemas operativos, requisitos de memoria, prioridades en los laboratorios, etc.

Apéndice A. Conexión de laboratorios mediante el mecanismo Flocking.

Para finalizar con los objetivos que se fijaron al principio del proyecto se hace necesario el uso de un mecanismo que permita la comunicación entre los diferentes laboratorios del área de Ingeniería Telemática, con el fin de que, al lanzar una tarea, sea posible usar el total del poder de procesamiento que se tiene en los laboratorios del área y cuyos resultados se presentaron en la introducción de este proyecto final de carrera.

Si en cada laboratorio se supone que hay configurado un Condor Pool, habrá al menos un Central Manager, por ejemplo el servidor de cada laboratorio, y mediante la herramienta de Flocking se puede crear un entorno Grid que permita la comunicación entre los diferentes laboratorios. El éxito de Grid reside en la utilización de recursos que se encuentran dentro de diferentes dominios administrativos. Debido a la rápida evolución de los competidores, Condor posee su propio mecanismo nativo, para realizar cálculos Grid, así como para llevar a cabo interacciones con otros sistemas Grid.

Flocking es un mecanismo nativo que permite que las tareas lanzadas en Condor desde un Pool, sea ejecutada en otro Pool diferente. Flocking debe de ser activado a través del archivo de configuración en cada uno de los Pool, que en este caso es cada uno de los laboratorios del área de Telemática. Una ventaja de Flocking es que las tareas migran desde un Pool a otro basándose en la capacidad que posean las máquinas del Pool para ejecutar las tareas. Cuando el Pool local no está capacitado para ejecutar la tarea (porque por ejemplo todas las máquinas estén incapacitadas u ocupadas), la tarea es enviada a otro Pool. Otra ventaja del uso de Flocking es que el usuario no es consciente de ello, puesto que el fichero de descripción submit es independiente del mecanismo de flocking.

Otras formas de cálculo usando Grid son habilitadas mediante el uso del universo Grid, y más aún mediante la especificación grid_type. Para cualquier tarea de Condor, la tarea es lanzada en una máquina del Pool local de Condor. La localización donde se ejecuta es identificada como máquina remota, o como recurso remoto. Todos estos mecanismos Grid que Condor ofrece, se distinguen por el software ejecutado en el recurso remoto.

Conectando los Pools de Condor mediante Flocking

Flocking es la manera en la que Condor permite que las tareas que no pueden ejecutarse inmediatamente en el Pool donde fue lanzada, puedan ejecutarse en un Pool diferente. Las variables del fichero de configuración permiten que el demonio condor_schedd implemente Flocking.

Configurar Flocking

La configuración de Flocking es bastante sencilla, utilizando pocas variables de configuración. Si se desea que las tareas de una máquina A, puedan ser ejecutadas en otro Pool B, entonces en la configuración hay que tener en cuenta las siguientes variables:

FLOCK_TO

Posee una lista separada por comas de los Central Manager de los Pools, a donde las tareas de la máquina A pueden ir para ser ejecutadas.

FLOCK_COLLECTOR_HOSTS

Es la lista de los demonios condor_collector dentro de los Pools que las tareas de la máquina A pueden mandar. En muchas ocasiones, es la misma que FLOCK_TO y puede ser definida de la siguiente manera:

FLOCK_COLLECTOR_HOSTS= $(FLOCK_TO)

FLOCK_NEGOTIATOR_HOSTS

Es la lista de los demonios condor_negotiator dentro de los Pools que las tareas de la máquina A pueden migrar. En muchos casos es la misma que la que se define en FLOCK_TO y se puede definir como:

FLOCK_NEGOTIATOR_HOSTS= $(FLOCK_TO)

HOSTALLOW_NEGOTIATOR_SCHEDD

Provee un nivel de acceso basado en host y una lista de autorización para el demonio condor_schedd que permite la negociación (por razones de seguridad) desde la máquina A hacia las máquinas de los Pools donde las tareas de la máquina A pueden migrar. Su valor suele ser el que se da por defecto:

HOSTALLOW_NEGOTIATOR_SCHEDD=$(COLLECTOR_HOST),
$(FLOCK_NEGOTIATOR_HOST)

Las macros de configuración que deben de usarse en el Pool de destino B son las que autorizan a las tareas de la máquina A migrar hacia el Pool B. Estas macros, no son más que una lista de máquinas separadas por comas (FLOCK_FROM), hacia donde las tareas pueden migrar.

HOSTALLOW_WRITE_COLLECTOR = $(HOSTALLOW_WRITE), $(FLOCK_FROM)

HOSTALLOW_ WRITE_STARTD = $(HOSTALLOW_WRITE), $(FLOCK_FROM)

HOSTALLOW_READ _COLLECTOR = $(HOSTALLOW_WRITE), $(FLOCK_FROM)

HOSTALLOW_READ_STARTD = $(HOSTALLOW_WRITE), $(FLOCK_FROM)

Caso particular de los laboratorios de la universidad

En el caso particular de la universidad hay nueve laboratorios, teniendo todos salvo el I+D+I-1 y el I+D+I-3, su propio servidor. Por ello el primer paso consistiría en la instalación de un servidor en cada uno de los laboratorios, para de esa manera poder emplearlos.

El siguiente paso sería la asignación de una IP estática a cada uno de los servidores de los laboratorios que actuarán como Central Manager como por ejemplo los que se muestran en la siguiente tabla:

IT-1	IT-2	IT-3	IT-4	IT-5	IT-6	I+D+I-1	I+D+I-2	I+D+I-3
192.168.1.1	192.168.1.2	192.168.1.3	192.168.1.4	192.168.1.5	192.168.1.6	192.168.1.7	192.168.1.8	192.168.1.9

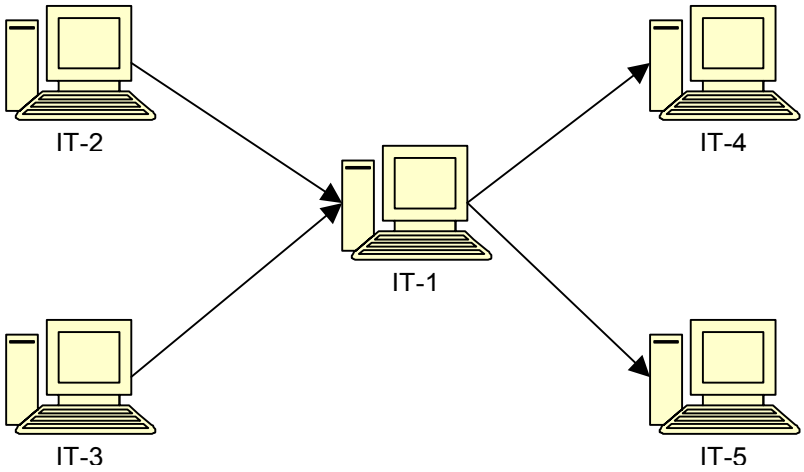
Apéndice 1. 1 Asignación IP estática en servidores AIT

Hay que indicar que estos servidores disponen de una IP estática pública, que también podría utilizarse para tales efectos.

El siguiente paso sería indicar al Central Manager de cada laboratorio hacia donde permite hacer el mecanismo de Flocking, y hacia donde permite realizar Flocking. Por ejemplo, si se quisiera que el laboratorio IT-1 recibiera tareas del IT-2 y el IT-3, y enviara tareas al IT-5 y al IT-6, teniendo sólo el Central Manager del IT-1 los permisos de administrador, y dejando que todas las máquinas tuvieran permisos de lectura y escritura, la parte de Flocking del archivo de configuración global sería:

```
## Flocking: Submitting jobs to more than one pool
## Flocking allows you to run your jobs in other pools, or lets
## others run jobs in your pool.
## To let others flock to you, define FLOCK_FROM.
## To flock to others, define FLOCK_TO.
FLOCK_FROM = 192.168.1.2, 192.168.1.3
FLOCK_TO = 192.168.1.4, 192.168.1.5
FLOCK_NEGOTIATOR_HOSTS = $(FLOCK_TO)
FLOCK_COLLECTOR_HOSTS = $(FLOCK_TO)
ºHOSTALLOW_ADMINISTRATOR = $(FULL_HOSTNAME)
HOSTALLOW_OWNER = $(FULL_HOSTNAME), $(HOSTALLOW_ADMINISTRATOR)
HOSTALLOW_READ = *
HOSTALLOW_WRITE = *
## Negotiator access.  Machines listed here are trusted central
## managers.  You should normally not have to change this.
HOSTALLOW_NEGOTIATOR = $(CONDOR_HOST)
## Now, with flocking we need to let the SCHEDD trust the other
## negotiators we are flocking with as well.  You should normally
## not have to change this either.
HOSTALLOW_NEGOTIATOR_SCHEDD = $(CONDOR_HOST), $(FLOCK_NEGOTIATOR_HOSTS)
## Flocking Configs.  These are the real things that Condor looks at,
## but we set them from the FLOCK_FROM/TO macros above.  It is safe
## to leave these unchanged.
HOSTALLOW_WRITE_COLLECTOR = $(HOSTALLOW_WRITE), $(FLOCK_FROM)
HOSTALLOW_WRITE_STARTD    = $(HOSTALLOW_WRITE), $(FLOCK_FROM)
HOSTALLOW_READ_COLLECTOR  = $(HOSTALLOW_READ), $(FLOCK_FROM)
HOSTALLOW_READ_STARTD     = $(HOSTALLOW_READ), $(FLOCK_FROM)
```

El esquema de conexión de la red de laboratorios creada sería la siguiente:



Apéndice 1. 2 Ejemplo de red creada mediante Flocking

Por ello, para realizar una conexión completa de todos los laboratorios entre sí, sólo sería necesario en FLOCK_FROM y en FLOCK_TO, escribir una lista completa de todos los Central Manager, dejando que cada Central Manager de manera individual manejara las cuestiones administrativas.

Bibliografía

La información utilizada en este proyecto, se ha obtenido de diferentes páginas Web y documentos obtenidos de las páginas oficiales de los entornos de computación y páginas donde hacen referencias a dichos entornos, como ha continuación se redactan:

Paginas Oficiales de los entornos de computación:

- OpenMosix: http://openmosix.sourceforge.net/
- Beowulf: http://www.beowulf.org/overview/index.html
- Ibis: http://www.cs.vu.nl/ibis/
- Globus Toolkit: http://www.globus.org/toolkit/
- Vgrid: http://vgrid.sourceforge.net/index.html
- OurGrid: http://www.ourgrid.org/
- Condor: http://www.cs.wisc.edu/condor/

Paginas de información general:

- Centro informático científico de Andalucía: http://www.cica.es/herramientas-cientificas-open-source.html
- Wikipedia (GRID): http://es.wikipedia.org/wiki/Grid
- Wikipedia (CLUSTER): http://es.wikipedia.org/wiki/Cluster_de_computadores
- Wikipedia (Superordenador): http://es.wikipedia.org/wiki/Supercomputadora

Libros de interés:

- Ian Foster, Carl Kesselman (1999). La Rejilla: libro azul para una nueva infraestructura informática (The Grid: Blueprint for a New Computing Infrastructure). Morgan Kaufmann Publishers. ISBN.
- Fran Berman, Anthony J.G. Hey, Geoffrey Fox (2003). La rejilla informática: haciendo realidad la Infraestructura Global (Grid Computing: Making The Global Infrastructure a Reality). Wiley. ISBN.

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

"Estudio, implementación y evaluación de entornos de computación de alto rendimiento HTC"

AUTOR: María Rosa Liarte López
DIRECTOR: Mª Victoria Bueno Delgado

Diciembre / 2007

Autor	María Rosa Liarte López
E-mail del Autor	rosarosike@hotmail.com
Director(es)	Mª Victoria Bueno Delgado
E-mail del Director	Mvictoria.bueno@upct.es
Codirector(es)	
Título del PFC	"Estudio, implementación y evaluación de entornos de computación de alto rendimiento HTC"
Descriptores	

Resumen

Para llevar a cabo el procesamiento de tareas con un cómputo muy grande se suele recurrir a lo que hoy día llamamos "supercomputadoras" que, con una capacidad de procesamiento muy superior a la de los ordenadores de sobremesa, son capaces de ejecutar cientos de tareas en un tiempo relativamente corto. El principal problema de estos "superordenadores" es el coste económico, ya que es este hardware presenta un precio muy elevado que no todas las empresas pueden abordar. Para solventar el obstáculo económico, en los últimos años han comenzado a utilizarse soluciones más rentables: los llamados entornos de alto rendimiento computacional HTC (High-Throughput Computing).

Tras comprobar como un entorno HTC permite aprovechar los recursos de los que dispone una red cuando se detecta inactividad en ésta, se describe en este proyecto la implementación de un entorno HTC prototipo para una futura implementación en los laboratorios de Área de Ingeniería Telemática. Este entorno permitirá al personal docente e investigador utilizar los recursos de los que dispone el área para poder realizar sus tareas de investigación de una forma sencilla y cómoda a través de una interfaz web realizada en un proyecto paralelo a este.

Titulación	Ingeniería Técnica de Telecomunicaciones, especialidad Telemática
Intensificación	
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Diciembre 2007

Índice

Índice figuras y tablas

Capítulo 1
Introducción

1.1 Antecedentes

Hoy en día, la mayoría de las empresas dedicadas a I+D+I así como las universidades necesitan simular y ejecutar numerosas tareas con el fin de obtener ciertos resultados. Algunas de estas tareas pueden consumir millones de ciclos de computación, lo que implica tiempos muy elevados de ejecución. Para solventar este problema se recurre desde hace años a las llamadas "supercomputadoras" que, con una capacidad de procesamiento muy superior a la de los ordenadores de sobremesa, son capaces de ejecutar cientos de tareas en un tiempo relativamente corto. El principal problema de estos "superordenadores" es el coste económico, ya que es este hardware presenta un precio muy elevado que no todas las empresas pueden abordar.

Para solventar el obstáculo económico, en los últimos años han comenzado a utilizarse soluciones más rentables: los llamados entornos de alto rendimiento computacional HTC (*High-Throughput Computing*). Estos entornos están formados principalmente por una gran cantidad de ordenadores de sobremesa puestos en red, con el mismo o distinto sistema operativo. El objetivo de estos entornos es aprovechar los ciclos de procesamiento de los ordenadores cuando no se detecta actividad en ellos para poder ejecutar tareas previamente lanzadas por un usuario a un ordenador central de ese entorno. Este ordenador central se encarga de monitorizar la actividad del resto de ordenadores repartiendo las tareas que le llegan entre los ordenadores que no presentan actividad durante un periodo de tiempo. De esta forma se aprovechan los ciclos de computación de aquellas redes que, con los ordenadores encendidos, no presentan ninguna actividad, para poder utilizarlos emulando una "supercomputadora". Se deduce pues que la solución de un entorno distribuido HTC no es una tarea fácil de implementar debido a su complejidad a nivel software pero conlleva un gran ahorro económico.

La Universidad Politécnica de Cartagena, y más concretamente en el Área de Ingeniería Telemática dispone hoy en día de varias "supercomputadoras" o centros de procesamiento para poder lanzar las tareas que realiza el personal docente e investigador de una forma sencilla contribuyendo, por un lado, a la rápida respuesta de estas máquinas en tiempo de procesamiento y evitando así que el investigador gaste los recursos de su ordenador personal pudiendo, de esa forma, utilizarlos para otros fines. Algunos aspectos técnicos de estas supercomputadoras se detallan a continuación:

- "Sísifo", (sísifo.upct.es) máquina del Área de Ingeniería Telemática, está compuesta por una máquina Sun, modelo v40z con 4 procesadores de 2 núcleos AMD, modelo 880-885, a 2,6 GHz. El rendimiento teórico de esta máquina es de 38,4 Gflops.
- "Prometeo", (prometeo.sait.upct.es) máquina del Servicio de Apoyo a la Investigación Tecnológica (SAIT), es un sistema paralelo de memoria distribuida con un total de 16 procesadores a 1GHz, con 8 MB de caché

de nivel 2, 16Gbytes de memoria y unos 300Gbytes de almacenamiento en disco. Su rendimiento teórico es de 32Gflops.

Hay que destacar que la máquina de Prometeo no es de uso exclusivo del área, sino que la utilizan todos los investigadores de la Universidad Politécnica de Cartagena, lo que implica compartir los recursos de dicha máquina, algo que, en ocasiones, puede dar como resultado una larga espera a la hora de obtener resultados de las tareas lanzadas en esta máquina.

Por otro lado, el Área de Ingeniería Telemática dispone de seis laboratorios docentes y tres laboratorios de investigación (I+D) donde se contabilizan un total de 101 máquinas entre servidores, routers y PCs. En la siguiente tabla se muestra las características técnicas de estas máquinas en términos de procesamiento. Además se detalla un estudio realizado sobre el número medio de horas a la semana que se utilizan estos laboratorios.

Laboratorio	Nº Servidores / Router	Procesador Servidor / Router	Nº PCs	Procesador PCs	Nº horas uso a la semana	% uso a la semana	Nº horas libre a la semana	% tiempo libre a la semana
IT-1	1	450 MHZ	15	Core 2 duo 2,13 GHZ	10	5.95%	158	94.04%
IT-2	1	Core 2 duo 2,13 GHZ	15	Core 2 duo 2,13 GHZ	14	8.33%	154	91.66%
IT-3	1	2,4 GHZ	10	2,4 GHZ	21	12.5%	147	87.5%
IT-4	1	1,5 GHZ	7 2	1 GHZ 2,4 GHZ	8	4.76%	160	95.23%
IT-5	1	Core 2 duo 2,13 GHZ	10	Core 2 duo 2,13 GHZ	14	8.33%	154	91.66%
IT-6	1 1	2 Xeon 3,8 GHZ 2 UltraSPARC 2 400 MHZ	10	530 MHZ	18	10.71%	150	89.28%
I+d+I-1	X		1 5	Core 2 due 2,13 GHZ Core 2 Duo 500 MHZ	12	7.14%	156	92.85%
I+d+I-2	1	Dual Core 2,8 GHZ	7 1	3 GHZ 4,3 GHZ	12	7.14%	156	92.85%
I+d+I-3	X		2 4 1 1	Core 2 duo 2,13 GHZ Core 2 duo 1,86 GHZ 2,8 GHZ 2,4 GHZ	12	7.14%	156	92.85%

Tabla 1. 1 Estudio rendimiento computacional del AIT (I)

Realizando unos sencillos cálculos se puede obtener el rendimiento teórico en FLOPS (*Floating Operation Point per Second*) que ofrecen las 101 máquinas. Para ello hay que tener en cuenta que todos los procesadores de los laboratorios son de arquitectura Intel. Según la página oficial de Intel el rendimiento teórico de sus procesadores de un único núcleo (*Single Core*) es de 8 flops por hercio, mientras que los procesadores de doble núcleo (*Dual Core*) es de 10 flops por hercio. Según estas especificaciones se obtiene el siguiente rendimiento:

Procesador GHZ	Gflops Teóricos	PCs Single Core	PCs Dual Core	Gflops Totales
0,45	3,6	1	-	3,6
0,5	5	-	5	25
0,53	4,24	10	-	42,4
1	8	7	-	56
1,5	12	1	-	12
1,86	18,6	-	4	74,4
2,13	21,3	-	44	937,2
2,4	19,2	14	-	268,8
2,8	22,4/28	1	1	50,4
3	24	7	-	168
4,3	34,4	1	-	34,4
	Gflops Totales = 1672,2			

Tabla 1. 2 Estudio del rendimiento computacional del AIT (II)

Los cálculos que se han realizado no se han tenido en cuenta los servidores del laboratorio IT-6 ya que estos se utilizan para otros fines. Por tanto, se tienen un total de 99 máquinas que ofrecen un rendimiento computacional de 1672,2 Gflops.

Hay que tener en cuenta que estos PCs no están siempre disponibles, asi pues, a continuación se va a realizar una comparativa del rendimiento de los mismos frente a las dos "supercomputadoras" de las que dispone el área para conocer de una forma más exacta los ciclos de computación que el área está desaprovechando. Para ello, se asume un total de 12 horas diarias (de 9:00 p.m. a 9:00 a.m.) en las que los 99 PCs no están siendo utilizados. El número total de Operaciones de punto flotante que se podrían realizar durante únicamente una semana son:

Total FLOPS = $1672,2*10^9$

Total segundos en una semana (solo 12 horas al día) = 60*60*12*7 = 302400 s.

Nº operaciones totales a la semana (FLOP) = $\frac{1672,2*10^9}{302400} = 5,52*10^6$

Para el caso de prometeo, y suponiendo uso exclusivo solo para el Área de Ingeniería Telemática

Total FLOPS = $32*10^9$

Total segundos en una semana (solo 24 horas al día) = 60*60*24*7 = 604800 s.

Nº operaciones totales a la semana (FLOP) = $\frac{32*10^9}{640800} = 52,91*10^3$

Para el caso de Sísifo, el número de operaciones totales a la semana, obtenido siguiendo las mismos criterios que para prometeo son:

Nº operaciones totales a la semana (FLOP) = $\frac{38,4 * 10^9}{640800} = 59,92 * 10^3$

Tras la comparativa, se demuestra como la suma de los ciclos de computación de las máquinas que forman parte del Área de Ingeniería Telemática es muy superior al de las supercomputadoras que hoy por hoy pueden acceder los investigadores del área para lanzar sus tareas. Por tanto existe un claro desaprovechamiento de ciclos de computación de las máquinas de los laboratorios del área.

1.2 Objetivos

Tras comprobar como un entorno HTC permite aprovechar los recursos de los que dispone una red cuando se detecta inactividad en ésta, se plantea como objetivo realizar un entorno HTC prototipo para una futura implementación en los laboratorios de Área de Ingeniería Telemática. Este entorno permitirá al personal docente e investigador utilizar los recursos de los que dispone el área para poder realizar sus tareas de investigación de una forma sencilla y cómoda a través de una interfaz web realizada en un proyecto paralelo a este.

Para llevar a cabo este proyecto será necesario, primero, dar respuesta diversas cuestiones respecto a las necesidades del usuario final y solventarlas teniendo en cuenta las restricciones impuestas en cuanto al hardware y el software disponible en dichos laboratorios. A nivel instalación, se resolverán, entre otras, las siguientes cuestiones:

- ¿Qué tipo de entorno HTC se adapta más a las necesidades del prototipo “usuario final” y al hardware y software del que se dispone?
- ¿El sistema operativo que actualmente reside en los ordenadores de los laboratorios influye en dicho entorno?
- ¿Será necesario tener algún software adicional instalado en los ordenadores de los laboratorios para llevar a cabo la instalación del entorno?
- ¿La existencia de este entorno HTC en los laboratorios implica una reducción del rendimiento de los ordenadores cuando estos los utilicen los alumnos en su formación?
- ¿Se podría ampliar la instalación de este entorno al resto de laboratorios de la Escuela de Telecomunicaciones?

A nivel usuario, deberán responderse, entre otras, las siguientes preguntas:

- ¿El usuario necesitará realizar alguna instalación de software adicional en su ordenador personal para poder trabajar con el entorno?
- ¿Un usuario podrá acceder de forma simple y sencilla al entorno HTC?
- ¿Se podrán lanzar todo tipo de tareas, independientemente del lenguaje en el que hayan sido escritas?

Una vez solventadas esas cuestiones, el objetivo será la instalación, configuración y puesta en marcha del entorno adecuado realizando diversas pruebas para comprobar el perfecto funcionamiento del mismo y añadiendo funciones adicionales que se consideren útiles para un futuro próximo.

1.3 Estructura del contenido

El contenido del proyecto se define en los siguientes capítulos:

En el capítulo 2 se presentan los distintos entornos computacionales que se utilizan hoy en día y se justifica la elección del entorno computacional HTC Condor. Se describe el entorno, características principales, arquitectura y modos de funcionamiento, etc..

En el capítulo 3 se explican los pasos necesarios para la instalación de dicho entorno bajo los sistemas operativos Windows XP y Unix con la distribución RedHat. Se describe el escenario en el que se realiza la instalación así como los requisitos hardware y software. Por último se detalla la configuración necesaria para poner en marcha el entorno.

En el capítulo 4 se describen los procesos y tareas que se pueden lanzar en el entorno de computación instalado. También se describen los requisitos a cumplir y pasos a seguir para poder lanzar las tareas, para la ejecución y por último, la obtención de resultados.

En el capítulo 5 se detallan las conclusiones del trabajo realizado y las posibles líneas futuras.

Por último se añade un apéndice para explicar diversas configuraciones adicionales que se pueden implementar en el entorno para dar una mayor funcionalidad al mismo.

Capítulo 2
Entornos de Computación. La elección de Condor

2.1 Introducción a los entornos de computación.

Hace años, los científicos y los ingenieros dependían del uso de una máquina central, una supercomputadora, para que realizara el trabajo computacional. Un gran número de individuos y grupos necesitaban unir sus recursos financieros, para poder permitirse una de estas máquinas. Los usuarios debían de esperar su turno, y además tenían un espacio limitado de tiempo para llevar a cabo su tarea.

En el momento que los ordenadores se volvieron más pequeños, mas rápidos y baratos, los usuarios se distanciaron del uso de las supercomputadoras, y compraron sus propias estaciones de trabajo y PC´s. Un individuo o grupo pequeño podrían permitirse recursos computacionales suficientes para lo que se necesitase. “El PC es más pequeño que las supercomputadoras, pero posee acceso exclusivo”.

Para muchas investigaciones y proyectos de ingeniería, la calidad de las investigaciones o del producto es fuertemente dependiente con la cantidad de ciclos de ordenador que se permitan. Científicos e ingenieros que trabajan en esta clase de proyectos necesitan un entorno de ejecución que reparta gran capacidad de poder computacional a lo largo de un gran período de tiempo. Estos entornos de ejecución es lo que hoy en día se denominan “Entornos de computación”. Dentro de los entornos de computación hay que distinguir entre distintos tipos, los cuales se detallan en las siguientes secciones.

2.1.1 Entornos HPC

El término HPC (*High Performance Computing*) se refiere al uso de supercomputadoras (paralelas) y clústers de máquinas, esto es, sistemas computacionales formada por múltiples procesadores (normalmente de gran potencia), unidos en un sistema único, con conexiones comerciales disponibles. Esta arquitectura entra en contraste con las máquinas del tipo *Mainframe*, que son generalmente únicas por naturaleza. Mientras que tareas de cierto nivel necesitan usar este tipo de sistemas, pueden ser creados a partir de componentes separados. Debido a su flexibilidad, poder, y relativo bajo coste, los sistemas HPC han dominado de manera incremental el mundo de las supercomputadoras. Normalmente, los sistemas de ordenadores cercanos a la región de los tera-flops, son considerados ordenadores HPC.

Los sistemas HPC son mayoritariamente usados para investigaciones científicas. Un término relacionado HPTC (*High-Performance Technical Computing*), generalmente se refiere a aplicaciones del mundo de la ingeniería basadas en clúster (como los cálculos fluidomecánicos y el estudio virtual de los prototipos).

De manera reciente, HPC ha empezado a usarse para uso comercial, basado en clústers de supercomputadoras, como por ejemplo las data warehouse, aplicaciones line-on-business (LOB), y procesamiento de transacciones.

2.1.2 Entornos GRID

Desde que los ordenadores fueron conectados en red, la idea del Grid Computing ha estado latente, y no había progresado debido principalmente a la gran variedad técnica de la industria informática: múltiples sistemas operativos, arquitecturas de procesadores, lenguajes de programación, protocolos de red, etc. Sin embargo, debido a la perseverancia de sus seguidores, la omnipresencia de Internet y la casi ubicuidad de Windows, esta tecnología está haciéndose realidad.

El Grid Computing, es la tecnología que consta de una infraestructura que permite el acceso y procesamiento concurrente de un programa, entre varias entidades computacionales independientes, que actúan como un único gran sistema. Se usa normalmente para programas que requieren procesos de gran escala y/o acceso a mucha cantidad de datos.

Entre las características principales que distinguen al Grid Computing podemos citar las siguientes:

- Permite integrar sistemas y dispositivos heterogéneos, pues permiten que recursos diferentes puedan interactuar entre sí.
- Mejora del coste efectivo de los entornos operativos, pues permite aprovechar al máximo los recursos disponibles en una red, y de esta manera a su vez mejora la capacidad de los recursos para responder a las fluctuaciones de la demanda.
- Las tecnologías grid son flexibles, pues son capaces de ajustarse dinámicamente a los entornos cambiantes y fluctuantes de las tecnologías de la información.
- Aumenta la fiabilidad de la infraestructura, sacando ventaja de los recursos del grid como una alternativa ante la recuperación de los desastres tradicionales.

Los objetivos que persigue el Grid Computing para una empresa u organización se citan a continuación:

- Mejorar los tiempos para la producción: Pues permite incrementar la productividad y colaboración; y de esta manera las organizaciones mejoran sus tiempos de resultados y por lo tanto rapidez en el tiempo de lanzamiento al mercado, que en última instancia constituye una ventaja competitiva.
- Permitir la colaboración y promover flexibilidad operacional: Pues no solo unirá recursos tecnológicos dispares, sino también gente y aptitudes; permitiendo de esta manera la posibilidad de compartir, acceder y gestionar información, mejorando la colaboración entre unidades empresariales.
- Escalar para satisfacer demandas variables del negocio: Permite crear infraestructuras operativas flexibles y resistentes, que faciliten abordar rápidas fluctuaciones en la demanda, accediendo instantáneamente a recursos de computación y datos para "sentir y responder" a las necesidades de negocio.
- Incremento de la productividad: Dando a los usuarios finales acceso a los recursos de computación, datos y almacenamiento que necesiten y cuando

los necesiten, ayudando a las empresas a equipar mejor a sus empleados para efectuar sus tareas, resolver problemas comerciales complejos con facilidad y moverse entre etapas del diseño de productos, proyectos de investigación y más, todo más rápidamente.

- Aprovechar inversiones de capital existentes: Maximizar la utilización eficiente y productiva de los recursos existentes es una de las claves para reducir costes operativos. Además, las empresas pueden aprovechar los recursos grid para entregar escenarios de back up y recuperación efectivos y de bajo coste, sin necesidad de invertir para duplicar sistemas.

Sin embargo, y a pesar de las ventajas, el objetivo del proyecto es el uso optimizado de los recursos de los laboratorios de la facultad de telecomunicaciones, por lo que la principal característica de los entornos Grid, que es la capacidad de compartir recursos de diferentes zonas, se desaprovecha.

Además, y como se verá en las siguientes secciones, el entorno de computación elegido (HTC con la herramienta Condor) permite en su configuración el Universo Grid a partir del entorno HTC de Condor. Todo ello se realiza mediante el mecanismo de Flocking, cuyo funcionamiento se explica de forma más ampliada en el apéndice de este proyecto.

2.1.3 Entornos HTC

La clave de los entornos HTC (como Condor) es que aprovechan el uso de todos los recursos disponibles. Existen muchas ventajas para que se haya decantado por un entrono HTC, alguna de las cuales se citan a continuación:

- HPC puede ser usado para soportar decisiones o aplicaciones de tiempo limitado. Sin embargo para hacer análisis altamente sensibles, estudios o simulaciones que necesitan para su ejecución largos períodos de tiempo, se necesita un mecanismo como HTC.
- El software que opera con un mecanismo de alto rendimiento computacional HTC, en lugar de HPC, es capaz de organizar las máquinas en clústers o colecciones de clústers.
- Los entornos de computación HPC en ocasiones con descritos en términos de operaciones en coma flotante por segundo (FLOPS). Muchos científicos no tratan hoy en día las tareas FLOPS, ya que los problemas que esto trae consigo son de gran importancia. Las tareas en las que se centran los investigadores con aquellas que ocupan un largo período de tiempo de procesamiento para su resolución y un alto rendimiento de procesamiento, es decir, las tareas FLOPY (operaciones en coma flotante por año) bajo el entorno HTC.

2.2 Condor: entorno de computación masiva.

2.2.1 Introducción

Condor es un entorno de computación HTC, un “sistema conjunto” de máquinas que dirige tareas de gran cómputo. Como muchos sistemas conjuntos, Condor posee un mecanismo de colas, políticas de preferencia, esquemas de prioridad, y clasificaciones de los recursos. Los usuarios lanzan sus tareas en Condor, Condor pone la tarea en la cola, las ejecuta, e informa a los usuarios del resultado. Estos sistemas, normalmente operan sólo en máquinas dedicadas. Con frecuencia, estas

máquinas pertenecen a una organización, y sólo se dedican a la ejecución de tareas para dicha organización. Condor puede fijar tareas en máquinas dedicadas.

A pesar de los sistemas conjuntos normales, Condor también está diseñado para usar máquinas no dedicadas para ejecutar tareas. Indicando que sólo ejecuten tareas en máquinas que no están siendo usadas, Condor puede de manera efectiva usar máquinas que estén en espera en un conjunto (a partir de este momento se denominará Pool) de máquinas. Esto es importante, porque a menudo la capacidad de cómputo representado por la adicción total de todas las estaciones de trabajo no dedicadas a través de un Pool, es mucho mayor, que el poder de cómputo de un recurso central dedicado, como se pudo comprobar en la introducción de este proyecto.

2.2.2 Características principales

La funcionalidad básica de Condor es la siguiente: un usuario lanza una tarea sobre Condor. Condor busca una máquina adecuada en su red, y comienza a ejecutar la tarea en dicha máquina. ¿Y que pasa si por ejemplo, el dueño de la máquina donde se está ejecutando una tarea vuelve a su puesto de trabajo y comienza a utilizar el teclado?.Condor tiene la capacidad de detectar si una máquina que está ejecutando una tarea ya no puede hacerlo. Por tanto, puede migrar la tarea (checkpointing) hacia otra máquina que esté libre, continuando allí la ejecución, incluso si cambia el entorno de ejecución de Condor.

En aquellos casos donde Condor puede realizar checkpointing y migrar la tarea, Condor hace que resulte fácil el maximizar el número de máquinas que pueden ejecutar la tarea. En ese caso no hay requisitos para las máquinas de un sistema de ficheros distribuido (como por ejemplo NFS o AFS), por lo que las máquinas de toda una comunidad (un mismo Pool) pueden ejecutar una tarea, incluyendo las máquinas de diferentes dominios administrativos.

Condor puede ahorrar mucho tiempo, cuando una tarea debe de ejecutarse en muchas (cientos) de máquinas diferentes, con diferentes tipos de datos. Condor permite que con un solo comando se lancen todas las tareas. Dependiendo del número de máquinas que haya en el Pool, docenas o cientos de máquinas están preparadas para ejecutar la tarea en cualquier momento.

Condor no requiere tener una cuenta en las máquinas donde se ejecutan tareas. Condor puede hacer esto, gracias al sistema de llamada a procedimiento remoto (RPC) que toma las llamadas a las librerías para cada operación, como la lectura y escritura de ficheros. Estas llamadas son transmitidas a través de la red, para ser usadas por la máquina donde se ejecuta la tarea.

Condor implementa las ClassAds, un sencillo mecanismo que simplifica el lanzamiento de las tareas. Funciona de manera similar a los anuncios clasificados de un periódico. Todas las máquinas del Pool deben de indicar las propiedades de sus recursos, tanto los estáticos como los dinámicos. Un usuario especifica la solicitud de un recurso de ayuda cuando lanza una tarea. La fuente debe definir tanto los recursos necesitados, como los deseados para ejecutar tareas. Condor enlaza cada petición con su recurso correspondiente, para ejecutar la tarea. Durante este proceso Condor también considera los diferentes valores de prioridad.

Para terminar con esta introducción se explicarán algunas de las características especiales que posee Condor:

Migración y Checkpoint: cuando los programas pueden sen enlazados con las librerías de Condor, los usuarios de Condor pueden asegurarse de que las tareas sean completadas, incluso si se producen cambios en el entorno que Condor utiliza. Cuando

una máquina que ejecuta una tarea en Condor, se vuelve inutilizable, la tarea puede continuar, después de migrar a otra máquina.

No se necesitan hacer cambios en el código fuente de los usuarios: No se necesita una programación especial para usar Condor. Condor está preparado para ejecutar tareas no-interactivas. La migración y el checkpoint de los programas de Condor es transparente y automático, así como las llamadas a procedimiento remoto. Si se necesitan estas utilidades, el usuario sólo reenlaza las tareas. El código es recompilado, no cambiado.

Llamadas a procedimiento remoto: A pesar de ejecutar tareas en máquinas remotas, el modo de ejecución del universo Standard de Condor conserva el entorno de ejecución local a través de las llamadas a sistema remoto. Los usuarios no deben preocuparse de hacer que los ficheros de datos sean disponibles para las estaciones de trabajo remotas o también para obtener una cuenta de las estaciones remotas antes de que Condor ejecuten sus tareas allí. Los programas se comportan en Condor como si se estuvieran ejecutando en la estación de trabajo donde inicialmente se lanzaron sin importar en que máquina finalice realmente su ejecución.

Los Pools pueden trabajar de manera conjunta: Flocking es una característica de Condor que permiten que las tareas lanzadas en un Pool puedan ejecutarse en otro. El mecanismo es flexible, siguiendo las peticiones desde el lanzamiento de la tarea, mientras que son aceptadas por el segundo Pool, hasta el conjunto de máquinas de dicho Pool y se establecen las condiciones sobre las que las tareas se ejecutan.

Las tareas pueden ordenarse: El orden de ejecución necesario por las dependencias entre un conjunto de tareas puede establecerse de manera sencilla. El conjunto de las tareas emplea un grafo, donde cada nodo es una tarea. Las tareas son ejecutadas en Condor, siguiendo las dependencias dadas por el grafo.

Condor permite comportarse como un GRID: La técnica de glidein permite a las tareas lanzadas en Condor ser ejecutadas en máquinas grid en varias localizaciones alrededor del mundo.

Sensible a los deseos de los propietarios de las máquinas: El propietario de la máquina posee prioridad completa sobre el uso de la misma. El propietario generalmente permite la ejecución de las tareas de otros propietarios mientras que la máquina está en espera, pero desea que se termine su ejecución una vez que regrese. El usuario no debe hacer acciones especiales para recuperar el control. Condor realiza esta acción de manera automáticamente.

ClassAds: El mecanismo de ClassAd en Condor posee un marco muy flexible para unir las solicitudes de las fuentes con los recursos ofrecidos. Los usuarios pueden de manera sencilla solicitar los requisitos y los deseos de las tareas. El dueño de una estación de trabajo puede establecer una preferencia de modo que ejecute tareas de un conjunto específico de usuarios. El usuario también puede solicitar que no haya actividad entre las estaciones de trabajo a ciertas horas. Las preferencias, requisitos y las restricciones en la disponibilidad de los recursos, pueden se descritas en términos de expresiones potentes, permitiendo la adaptación de Condor de manera cercana, a cualquier política deseada.

2.2.3 Arquitectura y modos de funcionamiento

2.2.3.1 Funciones que una máquina puede desempeñar

Cada máquina en un Pool de Condor puede servir a una variedad de funciones. Muchas máquinas pueden desempeñar más de una función de manera simultánea. Ciertas funciones deben ser llevadas a cabo por máquinas únicas en el Pool. La

siguiente lista describe cuales son esas funciones, y que recursos con requeridos por la máquina que lleva a cabo el servicio:

Central Manager

Sólo puede haber un Central Manager en cada Pool. Esta máquina es la que recoge la información, y lleva a cabo la negociación entre las máquinas del Pool. Las dos responsabilidades del Central Manager son llevadas a cabo por demonios (procesos) separados, por lo que es posible tener máquinas diferentes dando los dos servicios. Sin embargo, normalmente se llevan a cabo por la misma máquina. Esta máquina toma una parte muy importante en el Pool de Condor, por lo que debe ser segura. Si esta máquina falla, no se podrán usar las utilidades de Condor. Por ello la máquina que sea el Central Manager debe mantenerse siempre activa, o en su defecto, debe ser aquella que pueda reiniciarse de la manera más rápida si algo sale mal. El Central Manager idealmente debe tener una buena conexión de red con todas las máquinas del Pool, ya que las máquinas deben de ser capaces de poner al día al Central Manager ante los posibles cambios. Todas las consultas van hacia el Central Manager.

Execute

Cualquier máquina del Pool (incluyendo el Central Manager) puede ser configurada para que pueda ejecutar o no tareas de Condor. De manera obvia, alguna de las máquinas debe realizar este papel, o el Pool no sería muy útil. Ser una máquina Execute no requiere mucho. La única característica es que tenga suficiente espacio de disco, lo suficiente para llevar a cabo la tarea, que será eliminada de la cola, una vez que haya mandado el resultado a la máquina que la lanzó. Sin embargo, si no se dispusiera de mucho espacio de disco, Condor simplemente limitaría el tamaño de las tareas que se pudieran lanzar. De manera general cuantos más recursos posea una máquina (memoria, velocidad de CPU..) más peticiones de las que le lleguen podrá servir. Sin embargo si son peticiones que no requieren muchas características, cualquier máquina del Pool podrá atenderlas.

Submit

Cualquier máquina del Pool (incluyendo el Central Manager) puede ser configurada, de manera que se la permita lanzar tareas de Condor. Las características de las peticiones para una máquina Submit van aumentando en comparación con los de las máquinas Execute. Lo primero de todo es que cualquier petición que se esté ejecutando en una máquina remota, genera otro proceso en la máquina Submit. Por ello si se tienen muchas tareas ejecutándose se necesita mucho espacio de memoria. Además todos los ficheros checkpoint de las tareas son almacenados en el espacio local que lanzó la tarea. Por lo tanto si la imagen de las tareas es muy grande, y se lanzan muchas de ellas, se necesitará mucho espacio de disco para almacenar esos ficheros. Los requisitos de espacio de disco pueden ser en cierto modo aliviados por un servidor Checkpoint, aunque los binarios de las tareas que se lanzan son guardados en la máquina Submit.

Servidor Checkpoint

Una máquina del Pool puede configurada como un Servidor Chekpoint. Esto es opcional, y no es parte de la distribución binaria estándar de Condor. El Servidor Checkpoint es una máquina centralizada que almacena todos los ficheros checkpoint de las tareas que se lanzan en el Pool.

Esta máquina debe poseer mucho espacio de disco y una buena conexión de red hacia el resto del Pool, puesto que el tráfico puede ser denso.

2.2.3.2 Procesos (Demonios) de Condor

La siguiente lista describe todos los demonios y programas que funcionan en Condor y la tarea que desempeñan:

condor_master

Este demonio es el responsable de mantener el resto de los demonios ejecutándose en cada máquina del Pool.

Genera el resto de los demonios, y periódicamente comprueba si hay nuevos binarios instalados para cada uno de ellos. Si los hay condor_master reiniciará el resto de los demonios. Además si algún demonio el demonio falla, el master mandará un correo al administrador del Pool de Condor y reiniciará el demonio. El master también soporta varios comandos administrativos que permiten iniciar, parar o reconfigurar los demonios de manera remota. A continuación se explican los más utilizados:

condor_on	Inicia parte de los demonios en las máquinas que se le indiquen. Este comando asume que el master se está ejecutando en cada máquina. Si no es el caso no podrá encontrar la dirección del master. Los demonios que se inicien son los que aparecen en la lista DAEMON_LIST del archivo de configuración
condor_off	Apaga los demonios de la lista DAEMON_LIST de las máquinas que se la indique. Esto lo hace de manera limpia por los que las tareas del tipo checkpoint puden terminar de manera exitosa con la mínima pérdida de trabajo.
condor_reconfig	Reconfigura todos los demonios de Condor en concordancia con el estado actual de los archivos de configuración. Hay algunas características de la configuración que sólo pueden renovarse si se usa este comando.
condor_restart	Reinicia los demonios de una serie de máquinas. El demonio será puesto en un estado consistente, eliminado y comenzado de nuevo.
condor_config_val	Sirve para poder ver la instalación actual de Condor en una máquina dada. Dada una lista de variables Condor indicará cual de esas variables está activada. También puede ser usada para cambiar el valor de determinadas características de los demonios de Condor. Ningún cambio tendrá efecto a menos que se ejecute condor_reconfig.

Tabla 2. 1 Comandos útiles del demonio master

condor_master se ejecutará en cada máquina del Pool, independientemente de la función que desempeñe cada máquina. Con cierta periocidad, se ejecutará el comando condor_preen para eliminar aquellos ficheros que ya no le sean necesarios a Condor.

condor_startd

Este demonio representa un recurso (máquina capaz de ejecutar tareas) dado del Pool. Advierte de ciertos atributos que debe tener la máquina Execute, dependiendo de las características de las tareas que se lanzan. Este demonio se ejecutará en cada máquina del Pool a la que se le permita ejecutar tareas. Es responsabilidad de los que lanzar las tareas determinar bajo que condiciones las tareas deben ser empezadas, suspendidas, etc. Cuando el demonio startd está preparado para ejecutar una tarea de Condor genera el demonio condor_starter.

También posee ciertos comandos administrativos. Uno de los más importantes es condor_vacate lo que permite a Condor realizar checkpoint ejecutándose en una serie de máquinas y permite a las tareas abandonar la máquina, aunque el trabajo permanecerá en la cola hasta que sea definitivamente eliminado. Si se posee máquinas con multiprocesadores, se tendrá un demonio condor_startd por procesador.

condor_starter

Este demonio es la entidad que actualmente genera la tarea de Condor en una la máquina remota. Inicia el entorno de ejecución y monitoriza la tarea una vez que se está ejecutando. Cuando se termina la tarea, el demonio starter lo notifica mandado la información de estado necesaria a la máquina Submit, y termina la ejecución.

condor_schedd

Este demonio representa la solicitud de recursos al Pool. Cualquier máquina que se desee que pueda lanzar tareas, necesitan tener el demonio condor_schedd ejecutándose.

Cuando un usuario lanza una tarea, esta se remite al demonio schedd, que la sitúa en la cola de tareas, la cual controla. Muchas herramientas que permiten manejar y visualizar la cola, deben remitirse a este demonio para realizar su trabajo :

condor_submit	Es el comando que se encarga para lanzar las tareas a través de Condor. Para ello se requiere un fichero de descripción submit que se comentará más adelante
condor_rm	Elimina las tareas de la cola
condor_q	Muestra las tareas que se encuentran en la cola.

Tabla 2. 2 Comandos útiles del demonio schedd

Si el demonio schedd se para, ninguna de estas funciones podrá llevarse a cabo. El demonio schedd controla el número de tareas en espera en la cola de trabajo, y es responsable de encontrar máquinas adecuadas para realizar las peticiones. Una vez que el schedd se ha asignado a un recurso, el schedd genera el demonio condor_shadow, que servirá a esa petición en particular.

condor_shadow

Este demonio se ejecuta en la máquina donde se lanzó la tarea y actúa como el manejador de los recursos para dicha petición. Las tareas que son enlazadas hacia el Universo Standard (el cual se explicará en la siguiente sección), realizan llamadas a procedimientos remotos a través del condor_shadow. Una llamada a sistema creada en la máquina Execute remota se envía a través de la red, hacia el condor_shadow que realizará la llamada del sistema (como las operaciones de entrada/salida) en la máquina que lanzó la tarea, y el resultado se envía de nuevo a través de la red a la tarea remota.

Además, este demonio es el responsable de tomar decisiones sobre las peticiones (como por ejemplo donde deben almacenarse los ficheros checkpoint, como debe de ser el acceso a ciertos archivos, etc.). Uno de los comandos más importantes que emplea es condor_status. Es una herramienta versátil que puede ser empleada para monitorizar y controlar el Pool. Puede usarse para consultar información de los recursos

condor_collector

Este demonio es el responsable de recolectar la información de estado de todo el Pool de Condor. El resto de los demonios, de manera periódica enviarán ClassAd para poner al día al collector. Estos ClassAds contienen información completa sobre el estado de los demonios, los recursos que representan o los recursos necesarios para las tareas del Pool (como tareas que deben de tener un determinado schedd). El comando condor_status puede ser usado para preguntar al collector acerca de información específica sobre varias partes de Condor. Asimismo los demonios de

Condor, por sí mismos preguntan al collector sobre información importante, como por ejemplo que dirección usar para enviar comandos a una máquina remota.

condor_negotiator

Este es el demonio responsable de todas las negociaciones que se llevan a cabo en el Pool. De manera periódica, este demonio comienza un ciclo de negociación, donde pregunta al collector sobre el estado actual de todos los recursos del Pool. Contacta con cada schedd que está esperando para satisfacer las peticiones en orden de prioridad, y trata de encontrar los recursos adecuados para satisfacer dichas peticiones. Este demonio es el responsable de hacer cumplir las prioridades del sistemas, donde cuantas más características un determinado usuario ha solicitado, se le otorgará menor prioridad en las próximas peticiones. Si un usuario con una mayor prioridad tiene tareas que están esperando para ejecutarse, y un usuario con una prioridad menor solicita recursos, el demonio negotiator puede adelantar dicho recurso y hacerlo de manera paralela a atender la petición del usuario con la mayor prioridad.

Nota: La prioridad de un usuario será mayor cuanto menor sea el valor numérico que tenga dicha prioridad (es decir, una prioridad de 0.5 es mejor que una prioridad de 1).

condor_kbdd

Este demonio solo se emplea en Digital Unix. En dichas plataformas condor_startd no es capaz de determinar la actividad de consola (como el ratón o el teclado) de manera directa por el sistema. El demonio kbdd conecta con el servidor X y comprueba de manera periódica si ha habido alguna actividad. Si la hay kbdd manda un comando al startd. De esa manera el demonio startd reconoce que el usuario está usando la máquina de nuevo y puede decidir que acciones son necesarias.

condor_ckpt_server

Éste es el servidor checkpoint. Da servicio a las tareas de almacenamiento y envío de ficheros checkpoint. Si el Pool ha sido configurado para usar un servidor checkpoint, pero esa máquina cae Condor vuelve a mandar los ficheros checkpoint de una determinada tarea de vuelta a la máquina que lanzó la tarea.

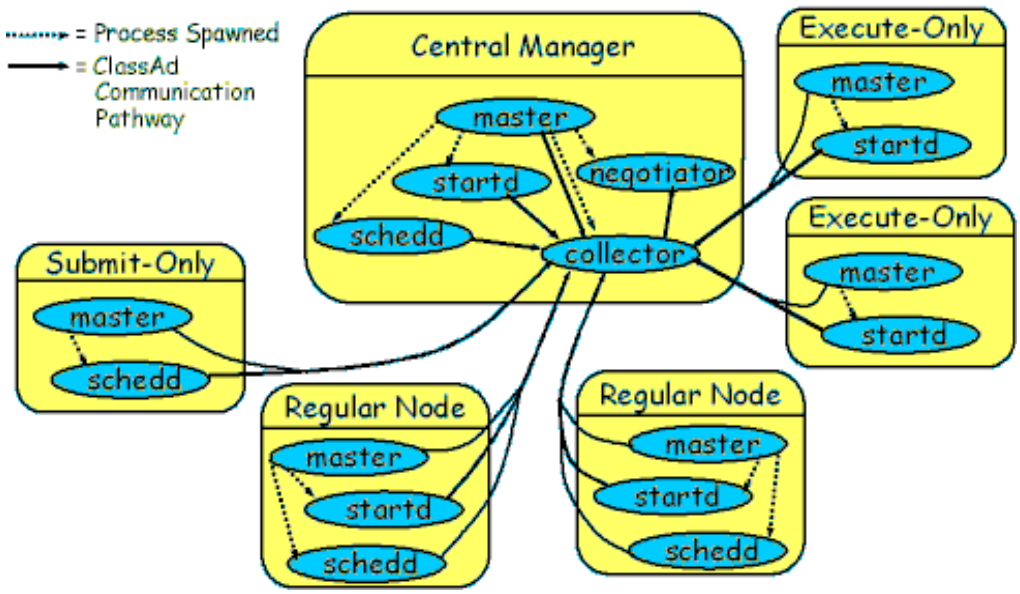


Figura 2. 1 Demonios de un Pool

2.2.3.3 Estados de las máquinas de un Pool.

Condor le asigna un estado a cada máquina. Este estado depende de si a la máquina se le permite lanzar tareas, y también de en que parte de la negociación se encuentre. Los posibles estados son:

Owner

El estado Owner representa a una máquina que está siendo usada por el propietario de la misma. La máquina permanece en este estado, mientras que la variable isOwner es true. Si la expresión isOwner se evalúa a false la máquina pasa al estado Unclaimed. El valor por defecto de esta expresión se selecciona dependiendo de cada Pool. La máquina permanecerá en el estado Owner tanto como la expresión START permanezca a false. Si la expresión START pone a true, o no se define, la transición de Owner a Unclaimed se produce.

Cuando se tienen máquinas dedicadas para el Pool, es decir, máquinas que sólo se encarguen de ejecutar tareas de Condor, se recomienda que la expresión isOwner se le asigne el valor false. El estado Unclaimed representa una máquina que no está siendo usada ni por el propietario de la misma, ni por el sistema Condor. Desde el punto de vista de Condor, apenas hay diferencia entre ambos estados. En ambos casos el recurso no está siendo usado por Condor. Sin embargo si una tarea solicita un recurso, la máquina está preparada para ejecutar la tarea, independientemente de que la máquina se encuentre en uno u otro estado.

La única diferencia entre uno u otro estado es de cómo el recurso es percibido por el demonio condor_status y otras herramientas de evaluación, y el hecho es que Condor no ejecutará benchmarking en un recurso que se encuentre en el estado Owner. Tan pronto como la expresión IsOwner se ponga a true la máquina volverá al estado Owner. Cuando la expresión se ponga a false, entrará en el estado Unclaimed. Un ejemplo de la expresión START es el siguiente:

START =KeyBoardIdle >15*$(MINUTE) || Owner == “Condor”

Esta expresión indica que cuando la máquina permanezca sin usar el teclado durante 15 minutos la máquina pasará del estado Owner al de Unclaimed. La expresión || indica que la máquina, aún estando en el estado Owner, si tiene tareas del usuario “Condor” tiene la posibilidad de llevarlas a cabo.

Mientras que la maquina está en el estado Owner, el demonio condor_startd comprobará el estado de la misma cada *UPDATE_INTERVAL*, para comprobar si algo ha cambiado, y la máquina debe ser llevada hacia otro estado. Esto minimiza el impacto sobre el usuario mientras que este está usando la máquina. La máquina sólo puede llegar al estado Unclaimed a través del estado Owner.

Unclaimed

Si la expresión isOwner se evalúa a true, entonces la máquina regresa al estado Owner. Si la expresión isOwner es false, entonces la máquina permanece en el estado Unclaimed. Si la expresión isOwner no aparece en los archivos de configuración, entonces el valor por defecto de esta expresión es:

START=?=FALSE

Por ello mientras que se encuentre en el estado Unclaimed, si la expresión START local se evalúa a false, la máquina regresa al estado Owner.

Cuando se encuentra en el estado Unclaimed, la expresión RunBenchmarks es importante. Si la expresión RunBenchmarks es true mientras que la máquina está en el estado Unclaimed, entonces la máquina pasa de la actividad Idle, a la actividad Benchmarking, y ejecuta benchmarks para calcular MIPS y KFLOPS. Cuando se completan las benchmarks, la máquina regresa al estado Idle. El demonio

condor_startd automáticamente inserta un atributo, LastBenchmark, cada vez que se ejecute benchmarks, por ello comúnmente Runbenchmarks se define en términos de de este atributo, por ejemplo:

```
BenchmarkTimer=(CurrentTime-LastBenchmark)
RunBenchmarks=$(BenchmarkTimer)>=(4*$(HOUR))
```

La expresión BenchmarkTimer calcula el tiempo que transcurre desde la última benchmark, por lo que cuando este tiempo execede de las 4 horas, se ejecutan benchmarks de nuevo. El demonio condor_startd guarda una media de estos tiempos, para seleccionar los recursos con los mejores tiempos

Matched

A la máquina se le permite ejecutar tareas, y ha sido escogido por el demonio negotiator con un schedd específico. Este schedd no ha reclamado aún esta máquina. En ese estado a la máquina no se le permite atender a más tareas. El estado Matched no es muy interesante para Condor. La máquina miente sobre su estado en la expresión START mientras que está en este estado, y establece la variable Requeriments a false para prevenir que sea seleccionada otra vez antes de que sea Claimed. Algo también interesante, es que el demonio condor_startd, empieza un temporizador, para asegurarse de que no permanece en este estado demasiado tiempo. Este temporizador se establece en la expresión MATCH_TIMEOUT del fichero de configuración. Se establece en segundos. Si el schedd al que se le asignó esta máquina, no la reclama después de este período de tiempo, la máquina regresa al estado Owner.

Claimed

La máquina ha sido solicitada por un schedd. El estado claimed es el más complejo de todos los estados. Tiene el mayor número posible de actividades, y el mayor número de expresiones que determinan sus próximas actividades. Además los comandos condor_checkpoint y condor_vacate afectan a la máquina cuando se encuentra en el estado Claimed. En general hay dos grupos de expresiones del archivo de configuración global que tienen efecto sobre este estado, dependiendo del tipo de Universo seleccionado. Por ejemplo:

```
WANT_SUSPEND=True
WANT_VACATE=$(ActivationTimer) > 10 * $(MINUTE)
SUSPEND=$(KeyboardBusy) || (CPUBusy)
```

Si se encontrase en el caso del Universo Vanilla, como es el caso de este proyecto, las expresiones poseen la cadena “_VANILLA” detrás de los nombres. Por ejemplo:

```
WANT_SUSPEND_VANILLA=True
WANT_VACATE_VANILLA=True
SUSPEND_VANILLA=$(KeyboardBusy) || (CPUBusy)
```

Si no se especifica el Universo Vanilla, Condor tomará el valor por defecto, que es el primero. El usuario debe especificarlo, ya que la principal diferencia estará en que las máquinas no realizaran checkpointing. Por ejemplo, un usuario podría desear, que las tareas del Universo Vanilla permanecieran más tiempo en espera, que las del Universo Standard. Mientras que la máquina se encuentra en el Universo Claimed, la variable POLLING_INTERVAL empieza a funcionar, y el demonio condor_startd, chequea la máquina más frecuentemente, para evaluar su estado. Si el propietario de la máquina empieza a usar de nuevo la consola, lo mejor es que Condor lo sepa lo antes posible, para ser capaz de empezar a hacer lo que el propietario de la máquina desee en dicho momento.

Hay una gran variedad de eventos, que pueden causar que el demonio condor_startd trate de deshacerse, o suspender de manera temporal un trabajo en ejecución. La actividad de consola de la máquina, descarga de archivos desde otras tareas, o el apagado del demonio condor_startd, por medio de un comando administrativo, pueden ser posibles causas de interferencias. Otra causa puede ser la aparición de una tarea con mayor prioridad, que quiera ejecutarse en la máquina. Dependiendo de la configuración, el demonio condor_startd puede actuar de una manera un poco diferente dependiendo de la máquina. El demonio condor_startd puede ser configurado para ignorar completamente la actividad de la consola, o para suspender la tarea, o también para eliminarla.

La expresión WANT_SUSPEND determina si la máquina debe evaluar la expresión SUSPEND, para considerar la completa suspensión de la tarea. La expresión WANT_VACATE, determina que ocurre cuando la máquina entra en el estado Premmpting. Este estado puede ir desde la suspensión, a la completa eliminación de la tarea. Si una o más de estas expresiones se pone a false, la máquina saltará de este estado eliminando la tarea, y procediendo directamente con los cambios más drásticos.

Si un comando condor_checkpoint se ejecuta, o la expresión PeriodicCheckpoint se evalúa a true, no hay ningún cambio de estado. El demonio condor_startd no tiene ninguna manera de conocer cuando se completa este proceso, por ello de manera periódica comprueba que no hay cambios.

Preempting

Este estado es menos complejo que el estado Claimed. Hay dos actividades. Dependiendo del valor de la expresión WANT_VACATE, una máquina se encuentra en la actividad Vacating (si está a True), o en la actividad Killing (si está a False). Mientras que se está en el estado Preempting, la máquina comprueba si la expresión Requirements es False, lo que significa que no está disponible para máquinas ajenas, por diferentes causas:

1) El propietario de la máquina la está utilizando
2) Otro usuario con una prioridad mayor tiene tareas esperando para ejecutarse.
3) Se ha encontrado otra solicitud que esta máquina debe servir antes.

Si la máquina se encuentra en la actividad Killing (ya sea porque la expresión WANT_VACATE es False, o porque la expresión KILL es True), se fuerza al condor_starter, para eliminar de manera inmediata la tarea. Una vez que la máquina ha comenzado a eliminar de manera forzosa la tarea, el demonio condor_startd comienza un temporizador, la duración del cual queda definido por la expresión KILLING_TIMEOUT (por defecto 30 segundos). Si este temporizador expira, la máquina actúa en la actividad Killing, algo puede ir realmente mal con el demonio condor_starter, y el condor_startd trata de eliminar la tarea, mediante el envío de la señal SIGKILL al condor_starter.

Una vez que el condor_starter ha eliminado todos los procesos asociados con la tarea, y una vez que se notifica al condor_schedd que la demanda no ha sido satisfecha, el estado de Preempting se abandona.

Backfill

Cuando una máquina está esperando a que vuelva el dueño de la máquina o a que se le asigne una tarea de Condor. Este estado sólo se produce si la máquina está especialmente configurada para admitir tareas backfill.

2.2.3.4 Actividades en los estados de una máquina.

Las actividades que se pueden tener dependiendo de los estados se enumeran a continuación:

Owner

Idle	Tan pronto como Condor se da cuenta que la máquina está en el estado Idle, Condor sabe que esta máquina no va a realizar ninguna actividad para Condor.

Unclaimed

Idle	La máquina permanece en este estado hasta que el usuario deja la máquina.
Benchmarking:	La máquina está ejecutando benchmarks para determinar la velocidad de la máquina. Esta actividad sólo ocurre en el estado Unclaimed. Cada cuanto realiza esta actividad se determina por la expresión *RunBencmarks*.

Matched

Idle	La máquina permanece en este estado hasta que el usuario la deja.

Claimed

Idle	la máquina ha sido reclamada por Condor, pero el schedd que la reclamó aún no ha llamado al condor_starter para que de servicio a la tarea. La máquina regresa a este estado (normalmente de manera breve) cuando las tareas (y por consiguiente el condor_starter) acaban..
Busy:	Una vez que el condor_starter empieza y el estado Claimed se activa, la máquina se mueve a la actividad Busy para indicar que está haciendo algo la máquina que le interesa a Condor.
Suspended:	Si el trabajo ha sido suspendido por Condor, la máquina entra dentro de la actividad Suspended. La relación entre la máquina y el schedd no se ha roto, pero el trabajo no está realizando ningún progreso y Condor no va a generar ninguna actividad en la máquina.
Retiring	Cuando una actividad es reclamada para ser cancelada por alguna razón mientras que espera a que se termine la tarea entra en este estado. La expresión MaxJobRetirementTime determina cuanto tiempo debe esperar. Una vez que el trabajo finaliza o el tiempo d espera se acaba, se entra en el estado Preempting.

Preeempting

Vacating:	En esta actividad la tarea que está ejecutándose está en el proceso de checkpointing. Tan pronto como el proceso de checkpointing se ha completado, el trabajo entra en el estado Owner o en el estado Claimed, dependiendo del motivo por el que se canceló.
Killing:	L la máquina ha solicitado que se elimine la tarea de manera inmediata, sin realizar checkpointing.

Backfill

Idle	La máquina ha sido configurada para que ejecute tareas backfill y está preparada para hacerlo, pero no ha tenido tiempo para crear un controlador backfill.
Busy	La máquina está realizando una ejecución backfill..

Killing	La máquina estaba realizando una ejecución backfill, pero ahora mismo está eliminando la tarea, bien porque el usuario lo ha solicitado, o para realizar una tarea normal de Condor..

2.2.3.5 Relaciones Estado-Actividad.

La siguiente figura trata de mostrar de manera gráfica las relaciones entre los diferentes estados, y las actividades que se pueden llevar a cabo en cada uno de ellos. Cada una de las flechas indica una transición de un estado a otro indicándose mediante colores el estado del que proceden, y con una letra el orden en el que se producen estas transiciones.

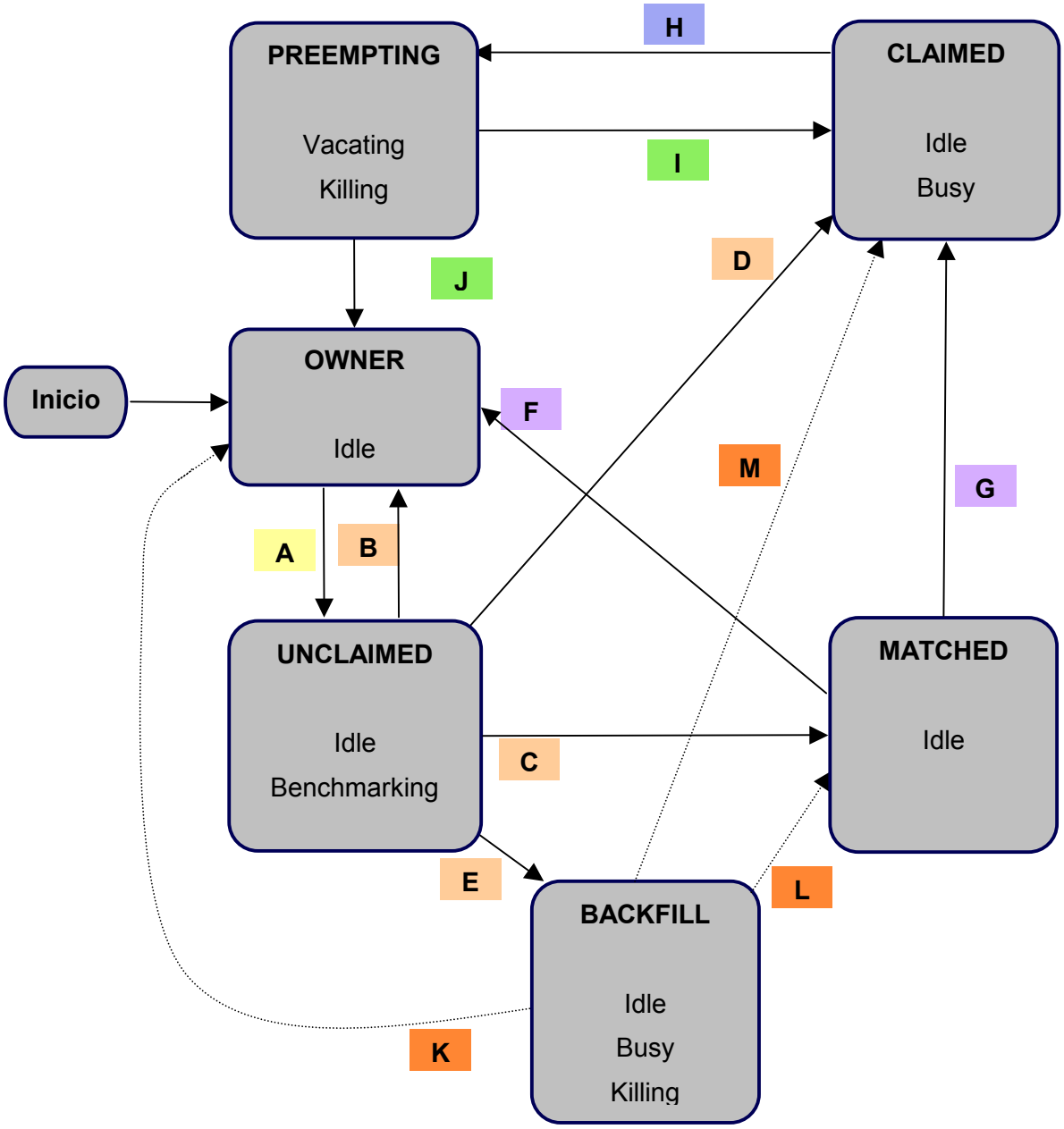


Figura 2. 2 Posibles estados y actividades de un Pool de Condor

- ***Transiciones desde el estado Owner***

  A. La máquina pasa del estado Owner al de Unclaimed, cada vez que la expresión START (del archivo de configuración) cambia a trae. Esto indica que la máquina está capacitada para ejecutar una tarea de Condor.

- ***Transiciones desde el estado Unclaimed***

  B. La máquina vuelve del estado Unclaimed al de Owner cada vez que la expresión START vuelve a tener el valor de false, eso significa que la máquina no está preparada para ejecutar tareas de Condor, porque está siendo usada por el propietario de la máquina.

  C. La transición desde el estado Unclaimed al de Matched ocurre cada vez que el demonio **condor_negotiator** le asigna una tarea.

  D. La transición desde el estado Unclaimed directamente al estado Claimed, también ocurre si el **condor_negotiator** le asigna una tarea. En este caso el **condor_schedd** recibe la asignación e inicia el protocolo claiming con la máquina, antes de que el **condor_startd** reciba la asignación de la tarea desde el **condor_negotiator**.

  E. La transición desde el estado Unclaimed al de Backfill ocurre si la máquina está configurada para ejecutar tareas Backfill, y la expresión START_BACKFILL se evalúa a trae.

- ***Transiciones desde el estado Matched***

  F. La máquina se mueve desde el estado Matched al de Owner si la expresión START vuelve a tener el valor de FALSE, o cada vez que la expresión MATCH_TIMEOUT expira. Este timeout se usa para asegurarse que si a una máquina el **condor_schedd** le asigna una tarea, pero ese **condor_schedd** no consigue contactar con el **condor_start** para llevarla a cabo, la máquina queda liberada de dicha petición, y está preparada para la siguiente asignación. En ese caso como la expresión START no ha sido evaluada a false, tan pronto como la transición **F** se completa, la máquina entrará inmediatamente en el estado Unclaimed de nuevo (a través de la transición **A**). La máquina también irá del estado Matched al de Owner si el **condor_schedd** trata de llevar a cabo el protocolo claiming pero ocurre alguna clase de error. Finamente la máquina si el demonio **condor_startd** recibe un comando **condor_vacate** mientras que se encuentra en el estado Matched.

  G. La transición desde el estado Matched al de Claimed ocurre cuando el **condor_schedd** completa de manera satisfactoria el protocolo claiming con el demonio **condor_startd**.

- ***Transiciones desde el estado Claimed***

  H. Desde el estado Claimed, el único destino posible es el estado Preempting. Esta transición puede ocurrir por diversas razones:

    - El demonio **condor_schedd** que ha reclamado la máquina no tiene más tareas que ofrecer.
    - La expresión PREEMPT se evalúa a true (lo que normalmente significa que el dueño dela máquina está haciendo uso de la misma).
    - El demonio **condor_startd** recibe el comando **condor_vacate**.
    - El demonio **condor_startd** se apaga (ya sea por una señal o por el comando **condor_off**).

- A la máquina se le asigna una tarea con una prioridad mayor (ya sea un usuario con una mayor prioridad, o que la expresión RANK de la máquina sea mejor).

➢ ***Transiciones desde el estado Preempting***

I. La máquina se moverá desde el estado Preeempting de regreso al de Claimed si se le asigna una tarea con una prioridad mayor.

J. La máquina se moverá del estado Preempting al de Owner si la expresión PREEMPT se evalúa a true, si se usa el comando **condor_vacate**, o si la expresión START se evalúa a false cuando el demonio **condor_startd** termina eliminando cualquier tarea que se estuviera ejecutando cuando entró en el estado Preempting

➢ ***Transiciones desde el estado Backfill***

K. La máquina se moverá del estado Backfill al de Owner por las siguientes razones:

- La expresión EVICT_BACKFILL se evalúa a true.
- El demonio **condor_startd** recibe el comando **condor_vacate**.
- El demonio **condor_startd** de apaga.

L. La transición desde el estado Backfill al de Matched ocurre cada vez que una máquina que ejecuta una tarea backfill es asignada por un demonio **condor_schedd** que desea ejecutar una tarea de Condor.

M. La transición desde el estado Backfill directamente hacia el estado Claimed es similar a la transición desde el estado Unclaimed directamente al estado Claimed. Sólo ocurre si el demonio **condor_schedd** completa el protocolo claiming antes de que el demonio **condor_schedd** reciba la notificación de la asignación de la tarea desde el demonio **condor_negotiator**.

2.2.3.6 Expresiones influyentes en las transiciones Estado/Actividad

Por último, y para terminar con este apartado sobre los estados, sus actividades y las transiciones entre los mismos, se va a realizar un resumen sobre las expresiones más influyentes, en los diferentes estados, de los archivos de configuración.

START: Cuando es True, la máquina está preparada para ejecutar tareas remotas.

Runbenchmarks: Mientras que se encuentra en el estado Unclaimed, la máquina deberá de ejecutar benchmarks cada vez que su valor sea true. En este proyecto ha sido necesario establecer el valor de esta variable a false, porque produce graves problemas a la hora de ejecutar tareas.

MATCH_TIMEOUT: Si la máquina ha permanecido en el estado Unclaimed más tiempo del que establece esta expresión, se producirá una transición al estado Owner.

WANT_SUSPEND: Si su valor es true, la maquina comprueba el valor de la expresión SUSPEND, para comprobar si debe suspender la tarea. Si su valor es false, comprobará la expresión PREEMPT.

SUSPEND: Si la expresión WANT_SUSPEND es true, y la máquina está en el estado Claimed, suspende la tarea si el valor de la variable SUSPEND es true.

CONTINUE: Si la máquina está en el estado Claimed, entra en la actividad Busy si la expresión CONTINUE es true.

PREEMPT: Si la máquina está en el estado Claimed, ejerciendo la actividad de Suspended o Busy, y la expresión WANT_SUSPEND es false, la máquina pasa a la actividad Retiring, si el valor de la variable preempt es true.

CLAIM_WORKLIFE: Si se especifica, esta expresión especifica el número de segundos durante los cuales un recurso continuará aceptando peticiones. Una vez que este temporizador expira, cualquier tarea existente, continuará ejecutándose de manera normal, pero una vez que la tarea termina, o queda en el estado preempting, el recurso deja de aceptar o ejecutar tareas. Esto es útil si se desea renegociar de manera periódica con las máquinas.

MAXJOBRETIREMENTTIME: Si la máquina se encuentra en el estado Claimed, en la actividad de Retiring, esta expresión especifica el número máximo de segundos, que el demonio condor_startd a la tarea, para terminar de manera natural (sin señales que fuercen su eliminación). El reloj se inicia cuando la tarea comienza y se para durante cualquier suspensión. La tarea debe ofrecer su propio MaxjobRetirementTime, pero sólo puede ser usada para que este tiempo sea menor que el proporcionado por el demonio condor_startd, nunca más. Una vez que la tarea acaba, o este tiempo expira, la máquina entra en el estado Preempting.

WANT_VACATE: Esta expresión solo se comprueba cuando la expresión PREEMPT es true y la máquina entra en el estado Preempting. Si WANT_VACATE es true, la máquina entra en la actividad Vacating. Si su valor es false, la máquina procederá directamente con la actividad Killing.

KILL: Si la máquina está en el estado Preempting, en la actividad de Vacating, entra en el estado Preempting, en la actividad de Killing, cuando el valor de esta variable sea true.

KILLING_TIMEOUT: Si la máquina está en el estado de Preempting, durante la actividad de Killing, y expira este contador, el demonio manda una señal SIGKILL al demonio condor_starter, para eliminar la tarea lo más pronto posible.

PERIODIC_CHECKPOINT: Si la máquina está en el estado Claimed, en la actividad Busy, y esta variable es true, la tarea ejecutará un checkpoint periódico

RANK: Si esta expresión se evalúa a un valor mayor por parte de una petición pendiente, que la que se está ejecutando en la máquina, la máquina pasa al estado Preempt la petición actual. Cuando se completa este estado, la máquina entra en el estado Claimed con la nueva petición.

START_BACKFILL: Cuando se encuentra a true, si la máquina está en la actividad de Idle, entrará en el estado Backfill, y realizará tareas Backfill (usando BOINC).

EVICT_BACKFILL

2.2.4 Universos

Los Universos de Condor definen los entornos de computación. El Universo debe especificarse en el fichero de descripción submit. Si no se especifica ningún Universo, Condor toma por defecto el Universo Standard. Los Universos soportados por esta versión son:

2.2.4.1 Universo Standard

En el Universo Standard Condor permite llamadas a procedimiento remoto y checkpointing. Estas características hacen que las tareas se ejecuten más fácilmente, y permite acceso remoto desde las fuentes hacia cualquier lugar en el Pool. Para preparar una tarea, como una tarea del Universo Standard, debe ser enlazada con condor_compile. Muchos programas pueden prepararse como tareas del Universo Standard, pero hay algunas restricciones.

Condor realiza el checkpoint de una tarea a intervalos regulares. Una imagen checkpoint es básicamente una captura del estado actual de una tarea. Si una tarea debe moverse de una máquina a otra, Condor realiza una imagen de la tarea, copia dicha imagen en la nueva máquina, y reinicia la tarea continuando por donde se había quedado. Si una máquina suele pararse o reiniciarse cuando ejecuta tareas, Condor puede reiniciar la tarea en una nueva máquina usando la imagen checkpoint más reciente que conserve. Gracias a esto las tareas pueden durar meses o años aunque la máquina ocasionalmente falle.

Las llamadas a procedimiento remoto se hacen de manera transparente al usuario, es decir, el usuario la percibe como si se estuvieran ejecutando en su propia máquina. Cuando una tarea se ejecuta en una máquina remota, un segundo proceso llamado condor_shadow corre en la máquina donde se lanzó la tarea.

Cuando la tarea atiende una llamada de sistema, condor_shadow realiza dicha llamada y manda los resultados a la máquina remota. Por ejemplo, si una tarea (en la máquina remota) necesita un fichero situado en la máquina que lanzó la tarea, condor_shadow buscará la tarea y enviará la información a la máquina donde se ejecuta la tarea.

Para convertir los programas en tareas del Universo Standard, se debe usar condor_compile para unir el programa a su correspondiente librería. Cuando se usa condor_compile no es necesario modificar el código fuente, pero se necesita tener acceso a los fichero *object*. Un programa comercial que está empaquetado en un solo fichero ejecutable no puede ser ejecutada por el Universo Standard. Por ejemplo si se desea enlazar el trabajo ejecutando

```
%    cc main.o tools.o -o pogram
```

entonces deberá relanzarse el trabajo para Condor con:

```
% condor_compile cc main.o tools.o –o program
```

Hay algunas restricciones sobre las tareas que pueden lanzarse en el Universo Standard:

1. Las tareas multi-procesador no están permitidas, esto incluye llamadas de sistema como fork(), exec(), and system().
2. No se permite la comunicación entre procesos.
3. Las comunicaciones en red deben ser cortas. –una tarea debe hacer conexiones de red usando llamadas de sistema, como son los sockets, pero las comunicaciones que permanezcan abiertas mucho tiempo retrasarán la migración y el checkpointing.
4. Enviar o recibir señales SIGUSR2 o SIGTSTP no está permitido. Condor reserva esas señales para su propio uso. Enviar o recibir otras señales si está permitido.
5. Alarmas, temporizadores, y señales de apagado no están permitidas. Esto incluye alarm(), getimer(), y sleep().

6. Los hilos múltiples a nivel de kernel no se permiten. Se permiten a nivel de usuario.
7. Los archivos de mapas de memoria no se permiten. Esto incluye las llamadas del tipo nmap() y munmap().
8. Los ficheros lock, pero no se guardan entre checkpoints.
9. Todos los ficheros deben abrirse como si fueran de “sólo lectura” o sólo escritura. Si los ficheros fueran de los dos tipos darían problemas a la hora de hacer las tareas de migración.
10. Si se realizan tareas de checkpoint se debe reservar una buena cantidad de espacio en disco, para almacenar las imágenes de las tareas que no han finalizado. Si el espacio de disco que se tiene es pequeño se puede designar una máquina para que haga de servidor Checkpoint, que almacenará todas las imágenes de las tareas.
11. En Linux las tareas deben ser enlazadas de manera estática. El enlace dinámico está permitido en las demás plataformas.

2.2.4.2 Universo Vanilla

El Universo Vanilla está diseñado para programas, que no pueden ser reenlazadas de manera satisfactoria. Los shell scripts son otro caso donde el Universo Vanilla es útil. Esto tiene consecuencias desafortunadas para las tareas que no son completadas en la máquina remota antes de que sean devueltas a su dueño. En este caso Condor sólo tiene dos opciones. Se pueden suspender las tareas, esperando a que se finalicen a posteriori, o puede comenzar el trabajo desde el principio en otra máquina del Pool. Como Condor no puede realizar llamadas a procedimiento remoto en el Universo Vanilla, el acceso a las entradas o salidas de las tareas se convierte en un problema. Una opción para Condor es depender de un sistema de ficheros distribuido como NFS o AFS. De manera alternativa, Condor tiene un mecanismo para transferir archivos en beneficio del usuario. En ese caso Condor transferirá todos los ficheros necesarios para realizar una tarea al lugar de ejecución, ejecutar la tarea, y transferir la salida de nuevo a la máquina que la lanzó.

En Unix Condor supone que se usa un sistema de ficheros distribuido para las tareas del Universo Vanilla. Sin embargo si no se dispone de dicho sistema un usuario puede activar el mecanismo de transmisión de ficheros de Condor. En las plataformas Windows, este mecanismo se usa por defecto.

2.2.4.3 Universo PVM

Este Universo permite a los programas escritos por la interfaz Parallel Virtual Machine ser usadas a través del entorno correspondiente de Condor.

2.2.4.4 Universo Grid

Es necesario proveer la interfaz de Condor a los usuarios que deseen lanzar tareas en sistemas administrados de manera remota.

2.2.4.5 Universo Java

Un programa del Universo Java debe poder ejecutarse en cualquier clase de máquina a pesar de su localización, propietario, o su versión JVM. Condor tendrá cuidado de todos los detalles, como buscar los binarios de JVM y establecer el classpath.

2.2.4.6 Universo Scheluder

Permite a los usuarios lanzar tareas pequeñas, para que sean ejecutadas de manera inmediata, a través del demonio **condor_schedd** por el mismo host Submit. Las tareas del Universo Scheluder no son emparejadas con una máquina remota, y nuncan pueden ser adelantadas. Estas tareas no obedecen la expresión *requirements*.

Originalmente creado para tareas del tipo meta-scheluder como condor_dagman, el Universo Scheluder también puede usarse para manejar tareas de cualquier clase que deban ejecutarse en las máquinas submit.

El Universo Scheluder no usa un demonio condor_starter para manejar la tarea, lo que limita las características y las políticas que soporta. El Universo Scheluder es la mejor opción que deben ejecutarse en la máquina Local, ya que ofrece una amplia variedad de características, y es más consistente que otros Universos, como puede ser el Vanilla. El Universo Scheluder puede retirarse posteriormente, a favor del nuevo Universo local.

2.2.4.7 Universo Paralelo

Permite la ejecución de tareas paralelas, como trabajos MPI, para que sea ejecutados a través del entorno adecuado de Condor.

2.2.4.8 Universo Local

Permite a las tareas de Condor ser lanzadas y ejecutadas con diferentes suposiciones sobre las condiciones de ejecución de la tarea. La tarea no debe esperar a que se le asigne una máquina, ya que se ejecuta en la máquina donde se lanzó la tarea. El trabajo no podrá adelantarse. Los requisitos de las máquinas no se consideran en el Universo Local.

Resumiendo, El Universo Standard permite la migración de las tareas, pero tiene algunas restricciones sobre las tareas que puede ejecutar. El Universo Vanilla permite pocos servicios, pero posee muy pocas restricciones. El Universo PVM es para programas inscritos a la interfaz Parallel Virtual Machine. El Universo MPI es para programas inscritos a la interfaz MPICH. El Universo MPI debe ser supervisado por el Universo Parallel. El Universo Gris permite a los usuarios lanzar tareas usando la interfaz de Condor. El Universo Java permite a los usuarios ejecutar tareas escritas para la máquina virtual de java (JVM). El Universo Scheluder permite lanzar tareas de pequeño tamaño manejadas por el demonio **condor_schedd** en el host donde se haya lanzado. El Universo Parallel es para programas que requieren múltiples máquinas para un mismo trabajo.

Capítulo 3
Instalación y puesta en marcha de Condor Pool

3.1 Pasos previos a la instalación

Antes de realizar la instalación, hay una serie de decisiones que se deben tomar, acerca del manejo de este Pool. Las decisiones responden a estas preguntas:

- **¿Qué máquina será el central manager?**

Una máquina del Pool debe de ser el Central Manager. Hay que instalar Condor en esta máquina primero. Éste será el núcleo de información central del Pool, así como la máquina que negociará entre máquinas que pueden ejecutar las tareas y las tareas que se lazan. Si el Central Manager falla, las tareas del sistema continuarán ejecutándose, pero ninguna otra podrá ser lanzada. Debido a la importancia de esta máquina para el correcto funcionamiento de Condor, el Central Manager debe ser una máquina que o bien nunca se apague, o bien sea capaz de reiniciarse de manera rápida si falla.

También es necesario considerar el tráfico de la red y el retardo de la misma cuando se selecciona el Central Manager. Todos los demonios mandan continuamente mensajes esta máquina. La necesidad de memoria en el Central Manager es mayor cuantas más máquinas haya en el Pool. Del mismo modo, una CPU más rápida mejorará el tiempo de negociación.

- **¿Qué máquinas podrán lanzar tareas?**

Condor puede restringir las máquinas que pueden lanzar tareas. Del mismo modo puede permitir a cualquier máquina de la red conectarse como una máquina Submit para poder lanzar tareas. Si el Pool de Condor se encuentra tras un firewall y todas las máquinas se encuentran detrás de dicho firewall, la configuración de entrada HOSTALLOW_WRITE debe establecerse con el valor *.

Además, es necesario reflejar el conjunto de máquinas a los que se les permite lanzar tareas. Condor trata de ser seguro, por ello, por defecto, no se permite a ninguna máquina lanzar tareas. Para evitar esta situación, basta con modificar en el fichero de configuración la entrada: YOU_MUST_CHANGE_THIS_INVALID_CONDOR_CONFIGURATION_VALUE

- **¿Debe Condor funcionar como Root o no?**

Se deben iniciar los demonios de Condor como usuario Root de Unix. Si no se inicia como Root Condor puede hacer muy poco en cuestiones de seguridad y políticas. Se puede instalar Condor como cualquier usuario, sin embargo esto conlleva serios problemas en seguridad y rendimiento.

- **¿Quién debe administrar Condor?**

Cualquier Root administrará Condor de manera directa, o alguien puede actuar como el administrador de Condor. Si el Root ha delegado la responsabilidad a otra persona, pero no desea darle a esa persona acceso de Root, el Root puede

especificar un fichero *condor_config.root* que permite sobrescribir acciones de los otros ficheros de configuración. De este modo, el fichero de configuración global *condor_config* puede ser controlado por quien sea condor_admin, y el *condor_config.root* sólo es controlado por el Root. Aquellos elementos que pueden comprometer la seguridad del Root (como los binarios lanzados como Root) pueden ser incluidos en el *condor_config.root* mientras que aquellos que sólo que se encargan de las políticas específicas de condor pueden ser controladas sin el acceso de Root.

- **¿Es necesario tener un usuario Condor de Unix? ¿Donde debe localizarse su directorio home?**

Para simplificar la instalación de Condor, crear un usuario Unix llamado Condor en todas las máquinas del Pool. Los demonios de Condor crearán ficheros (como los ficheros log) propios de cada usuario, y el directorio de Condor puede ser usado para especificar la localización de los ficheros y directorios necesarios para Condor. El directorio */home* de un usuario cualquiera puede ser compartido entre todas las máquinas del Pool, o puede ser un directorio */home* separado en la partición local de cada máquina. Ambas opciones tienen ventajas y desventajas. Tener los directorios centralizados hace que la administración sea más sencilla, pero también concentra el uso de recursos que potencialmente necesitan una gran cantidad de espacio, en un solo directorio */home* compartido.

Si no se crea un usuario llamado Condor de manera específica, entonces debe especificarse cualquiera que sea a través de la entrada de configuración CONDOR_IDS.

- **¿Dónde deben situarse los directorios específicos de Condor en la máquina?**

Condor necesita unos pocos directorios que son únicos en cada máquina del Pool. Dichos directorios son */sPool, /log* y */execute.* Generalmente estos tres directorios, son los subdirectorios del directorio local de la máquina (especificadas en la macro LOCAL_DIR en el fichero de configuración). Cada directorio local pertenece al usuario donde actúa Condor.

Si se posee un usuario Unix llamado Condor con un directorio */home* local en cada máquina, el LOCAL_DIR podría ser el directorio */home* de Condor (LOCAL_DIR=$(TILDE) en el fichero de configuración). Si lo que se tiene es un directorio */home* compartido entre todas las máquinas del Pool, se podría querer crear un directorio para cada host (indicado por su hostname) para el directorio local (por ejemplo, LOCAL_DIR=$(TILDE)/hosts/($HOSTNAME)). Si no se posee una cuenta específica de Condor en las máquinas, se pueden poner los directorios donde se desee:

Execute

Este es el directorio que actúa como el directorio de trabajo para cualquier tarea de Condor que se ejecute en una máquina Execute. El binario del trabajo remoto es copiado dentro de este directorio, por lo que debe haber suficiente espacio para ello (Condor no debe enviar una tarea a una máquina que no posea suficiente espacio en memoria para contener el binario inicial). De la misma manera, si el trabajo remoto es eliminado por cualquier razón, primero se elimina del directorio */execute*. Por ello es necesario poner el directorio */execute* en una partición con suficiente espacio de memoria.

Spool

Almacena la cola de trabajo y los ficheros históricos, así como los ficheros checkpoint de todas las tareas sometidas en dicha máquina. Como resultado los necesidades de espacio de disco pueden ser grandes,

especialmente si los usuarios lanzan tareas con ejecutables o tamaño de imagen muy grande. Usando un servidor Checkpoint, uno puede mejorar las necesidades de espacio, p.ej. todos los ficheros del Pool se almacenan en el servidor, en lugar de los directorios */sPool* de cada máquina. Sin embargo los ficheros checkpoint iniciales se almacenan en el directorio */sPool*, por lo que se necesita algo de espacio.

Log

Cada demonio de Condor escribe su propio fichero log, y cada fichero log está localizado en el directorio */log*. Se puede especificar que tamaño se desea para esos ficheros, ya que los requisitos del directorio son configurables. Si se posee un sistema de ficheros distribuido instalado en el Pool, se puede querer situar los directorios */log* en una dirección común (como /usr/local/condor/logs/$(HOSTNAME)), por lo que se pueden ver los ficheros log de todas las máquinas en una única dirección. Sin embargo, para ello se debe especificar una partición local para el directorio */lock*.

Lock

Condor usa un pequeño número de ficheros lock para sincronizar el acceso a determinados ficheros que son compartidos entre múltiples demonios. Debido a los problemas encontrados con el fichero locking y los sistemas de ficheros distribuidos, los ficheros lock deben de situarse en una partición local de cada máquina. Por defecto se encuentra en el directorio */log*. Si se sitúa el directorio */log* en una partición de un sistema de ficheros, especificar una partición para los ficheros lock con el parámetro LOCK del archivo de configuración.

Generalmente, se recomienda no establecer los directorios antes comentados (excepto el directorio */lock*) en la misma partición, como /var. Esto puede causar muchos problemas para las máquinas. Idealmente se debe tener una partición separada para los directorios Condor. Por tanto, la única consecuencia de no situar bien los directorios, es el mal funcionamiento de Condor, no de la máquina entera.

- **¿Dónde deben de situarse los diferentes componentes de Condor?**

-Ficheros de configuración

Hay una serie de ficheros de configuración que permiten diferentes niveles de control sobre como Condor está configurado en cada máquina del Pool. El fichero de configuración global está compartido por todas las máquinas del Pool. Por razones administrativas, este fichero debería estar localizado en un sistema de ficheros compartido si es posible. Además hay un fichero de configuración global para cada máquina, donde se pueden sobrescribir características del fichero global. Esto permite tener diferentes demonios activos, diferentes políticas sobre cuando empezar y parar Condor, y más aún. También se puede tener ficheros de configuración específicos para cada plataforma del Pool. También se recomienda empezar los demonios como Root; esto permite crear ficheros de configuración manejados por el Root, que sobrescribirán a los demás. De este modo, si el administrador de Condor no funciona como Root, los ficheros de configuración de Condor puede pertenecer y ser modificado por el administrador de Condor, pero el Root no tiene por qué garantizar al acceso de Root a esa persona.

En general, hay un número de lugares en el que Condor buscará los ficheros de configuración. El primero por el que buscará, será el fichero de configuración global. Las posibles localizaciones son descartadas en orden hasta que el fichero de configuración es encontrado. Si no contiene un archivo de configuración válido, Condor mostrará un mensaje de error y finalizará:

1. El fichero estará especificado en la variable de entorno CONDOR_CONFIG.
2. /etc/condor/condor_config
3. condor/condor_config
4. $(GLOBUS_LOCATION)/etc/condor_config

Si uno especifica un fichero en la variable de entorno CONDOR_CONFIG y hay un problema leyendo este fichero, Condor mostrará un mensaje de error y saldrá, sin continuar buscando en el resto de opciones. Sin embargo si no se especifica un fichero en la variable CONDOR_CONFIG, Condor buscará en las otras opciones.

A continuación Condor trata de cargar el fichero de configuración local. La única manera de especificar el fichero de configuración local, es en el fichero de configuración global, en la macro LOCAL_CONFIG_FILE. Si esa macro no está definida no se usa ningún fichero de configuración local. Esta macro puede se un único fichero o una lista.

La última en ser buscada es el archivo de configuración de Root. El fichero global es buscado en los siguientes sitios:

1 /etc/condor/condor_config.root

2 condor/condor_config.root

El fichero de configuración local de Root se encuentra en la macro LOCAL_ROOT_CONFIG_FILE. Si no se especifica esta variable este fichero no es usado. Esta variable puede ser un único fichero o una lista de los mismos

-Directorio Release

Cualquier distribución binaria contiene un fichero release.tar que contiene 5 subdirectorios: bin, etc, lib, sbin y libexec. Donde decida instalarse estos 5 directorios será el directorio realease (especificado en la macro RELEASE_DIR del fichero de configuración). Cada directorio release contiene unos binarios y librerías dependientes de la plataforma, por eso necesitas instalar una según cada máquina del Pool. Para facilitar la administración, esos directorios deben localizados en un sistema de archivos compartidos, si es posible.

-Binarios de usuario:

Todos los ficheros del directorio *bin* son programas que los usuarios de condor deben de tener en su ruta. Se pueden poner en una localización conocida (como /usr/local/condor/bin) la cual debe ser añadida por los usuarios Condor a su variable de entorno PATH, o copiar estos ficheros directamente dentro de un lugar conocido en el PATH de usuario (como usr/local/bin). También se pueden situar los binarios en /usr/local/bin haciendo enlaces simbólicos a /usr/local/bin.

-Binarios de sistema:

Todos los ficheros del directorio *sbin* son demonios de Condor y agentes, o programas que sólo el administrador de Condor necesita que funcionen. Hay que añadir esos programas sólo al PATH del administrador de Condor.

-Binarios privados de Condor:

Todos los ficheros del directorio *libexec* son programas Condor, que nunca deben se accionados manualmente, pero que son usados de manera interna por Condor.

-Directorio lib:

Los ficheros en el directorio lib son las librerías de Condor que deban ser lincadas, que serán usadas por las tareas migratorias y de checkpointing de los usuarios Condor. Lib también contiene escritos usados por **condor_compile** para ayudar a lincar los trabajos con las librerías de Condor. Estos ficheros deben estar en una localización que sea accesible para todo el mundo, pero no tiene por qué estar en el PATH de todos los usuarios. **Condor_compile** busca en el fichero de configuración para localizar el directorio lib.

-Directorio etc

Contiene un subdirectorio de ejemplos el cual contiene ejemplos de ficheros de configuración y otros ficheros usados para instalar Condor. Etc es la localización recomendada para guardar una copia master de los ficheros de configuración. Se pueden crear enlaces en cualquiera de las rutas anteriormente mencionadas sobre las que condor buscará automáticamente para buscar el fichero de configuración global.

-Documentación

La documentación dada por Condor, se puede encontrar en HTML, postscript y PDF. Se puede encontrar la documentación de condor en: http://www.cs.wisc.edu/condor/manual

- **¿Estoy usando AFS?**

Condor no posee una manera directa para autenticarse por si mismo a AFS. Esto supone que no se va a poder tener el LOCAL_DIR de condor en AFS. Sin embargo se puede tener el directorio RELEASE_DIR en AFS, con lo que se puede tener una copia de esos ficheros, y actualizarlos a través de una localización central.

- **¿Tengo suficiente espacio libre para Condor?**

Condor ocupa un espacio de disco considerable. Esta es otra razón por lo que es una buena idea tenerla en un sistema de archivos compartido. Las necesidades de espacio para las descargas, son dadas en la página de descargas. Además se debe tener suficiente espacio de disco en el directorio local de las máquinas que ejecutan tareas para Condor.

3.2 Condor Pool en S.O Windows.

3.2.1 Entorno del Condor Pool

Una vez explicadas las características más importantes de Condor, se procede a la instalación del mismo en una red prototipo.

Aunque la red que se muestra a continuación está compuesta por cuatro máquinas, la idea es que un futuro se proceda a su instalación en los laboratorios del área de ingeniería de telemática, para con ello aprovechar los recursos que hoy en día se están desaprovechando, como ya se explicó en la introducción de este proyecto.

Esta red prototipo está compuesta por cuatro máquinas y un hub, cuyas características se describen a continuación:

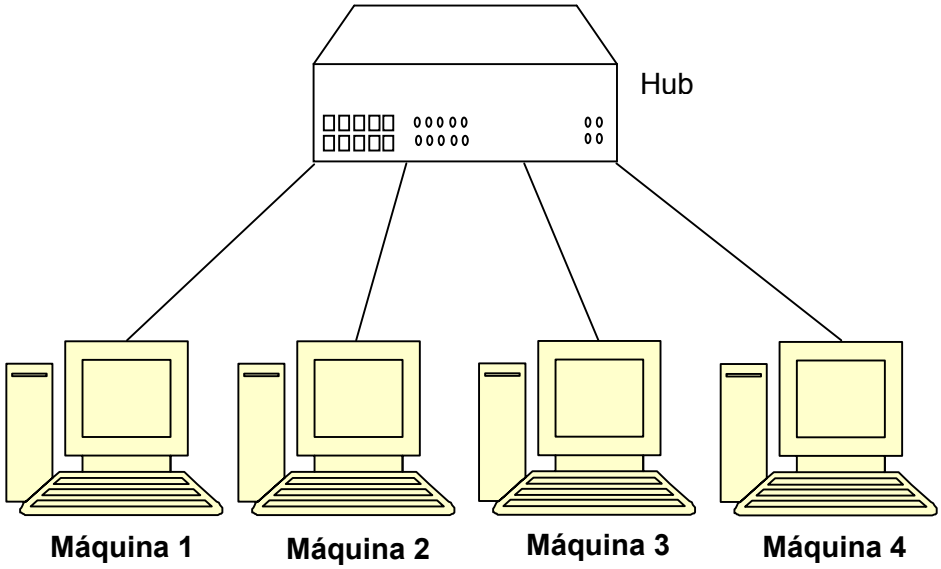


Figura 2. 3 Esquema de la red prototipo

Máquina	Dirección	Papel
1	192.168.1.1	Central Manager: execute (cada 15 minutos) y submit
2	192.168.1.2	execute (siempre) y submit
3	192.168.1.3	submit
4	192.168.1.4	execute y submit

Tabla 3. 1 Papeles que desempeñan las máquinas de la red prototipo

Se ha optado por la instalación del Universo Vanilla en todas las máquinas. El principal motivo de esta elección, fue el hecho de que las tareas del Universo Standard, al generar ficheros Checkpoint, necesitan una máquina (servidor Checkpoint), que se encargue del almacenamiento de esto ficheros. Esto supone un gran gasto de memoria, lo que a la larga hace que se desperdicien recursos de las máquinas. El Universo Vanilla, pese a no realizar checkpointing, es el más consistente de los posibles Universos. Ofrece fiabilidad y robustez. Las tareas, si se suspenden, permanecerán almacenadas en la máquina, y continuarán su ejecución cuando la máquina quede libre, sin perderse tiempo de ejecución. Además este universo es el más sencillo de adaptar a las posibles, que se deseen llevar a cabo en el Pool.

3.2.2 Requisitos Hardware

Las características hardware empleado en la realización del proyecto son las siguientes:

Función	Marca	Arquitectura	Memoria	Disco Duro
Central Manager,Submit y Execute	HP Pavilion	Intel x86	512 MB	80 GB
Submit	HP Pavilion	Intel x86	512 MB	80 GB
Execute	HP Pavilion	Intel x86	512 MB	80 GB
Execute	HP Pavilion	Intel x86	512 MB	80 GB

Tabla 3. 2 Características principales de las máquinas de la red prototipo

La arquitectura hardware de la red se basa en un Hub 3Com de 8 puertos, los ordenadores conectados a él a través de cables Ethernet directos categoría 5 UTP, toda la red a velocidad de 100 MB/s.

Cabe destacar que las características de este hardware en particular han limitado en gran medida, el software y las tareas que se pueden ejecutar.

3.2.3 Instalación del software Condor

Antes de la instalación es aconsejable crearse una nueva cuenta de Windows (en este caso condor con el password condor), para de esa manera tener almacenadas las tareas en un lugar distinto a nuestra cuenta habitual de Windows, y sólo iniciar el servicio cuando realmente se necesite. Si no se hace, será necesario asegurarse de que la cuenta posee una clave de acceso (y si no la posee crearla), pues en pasos posteriores se pedirá.

La instalación de Condor debe hacerse con privilegios de administrador. Después de la instalación, los servicios de Condor funcionarán en el sistema local. Cuando Condor ejecuta un trabajo lo ejecutará con los permisos de usuario.

El primer paso para realizar la instalación es descargar Condor de la página oficial: http://www.cs.wisc.edu/condor/downloads. Para poder proceder a la descarga, primero hay que registrarse. Una vez hecho esto, Condor remite a la página de descargas, donde se seleccionarán los binarios más adecuados para el hardware que se posea (indicado anteriormente). Cuando de haya descargado podrá comenzar el proceso de instalación haciendo doble clic en el icono que aparece tras la descarga. La instalación de Condor se completa resolviendo las preguntas, y escogiendo las opciones de los siguientes pasos:

Condor ya está instalado

Si Condor ya ha sido instalado en el ordenador, un cuadro de diálogo aparecerá antes del proceso de instalación de Condor. Se preguntará si se desean guardar, los ficheros actuales de configuración de Condor.

Figura 3. 1 Opciones de la instalación en Windows

Si se responde que si, los ficheros de configuración no se cambiarán, y el proceso continuará en el punto donde los nuevos binarios serán instalados. Si la respuesta es no, entonces se preguntará si se desean usar las respuestas dadas en la anterior instalación.

Paso 1: Acuerdo de licencia

El primer paso de la instalación es la bienvenida y el acuerdo de licencia. Se debe tener en cuenta que es mejor llevar a cabo la instalación, sin que ningún otro programa de Windows esté funcionando. Si se necesita cerrar algún programa, es más seguro cancelar el programa de instalación y cerrar el programa. Se debe aceptar la licencia. Se puede responder si o no. Si no se está de acuerdo con la licencia la instalación no continuará. Una vez que se acepte la licencia, la siguiente ventana de la instalación es para que se ponga el nombre e información de la compañía, o usar los que se dan por defecto.

Paso 2: Configuración del Condor Pool

La instalación de Condor requiere información, que depende de si el usuario está creando un nuevo Pool, o se une a un Pool existente. Si se está creando un nuevo Pool, la instalación requiere que la máquina donde se realice la instalación sea el Central Manager. Para la creación de un nuevo Pool, se preguntará sobre información acerca del nuevo Pool:

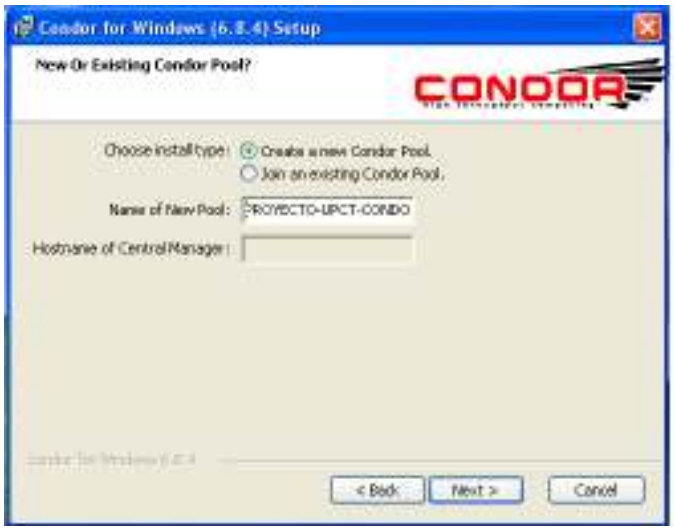


Figura 3. 2 Creación de un nuevo Pool en Windows

Si existe un Pool existente, el programa de instalación requiere el hostname del Central Manager. En este caso el central Manager se encuentra en la dirección ip 192.168.1.1:

Figura 3. 3 Unirse a un Pool existente en Windows

Paso 3: El papel de la máquina

Este paso es suprimido en la instalación de un Condor personal. Cada máquina de un Condor Pool puede lanzar tareas, o bien ejecutarlas. Este paso de la instalación permite elegir si la máquina sólo va a lanzar tareas (submit), sólo va a ejecutar tareas (execute), o ambas. El caso más común es ambos (el valor por defecto).En este caso los papeles de las diferentes máquinas son los siguientes:

Máquina	Dirección	Papel
1	192.168.1.1	Central Manager: execute (cada 15 minutos) y submit
2	192.168.1.2	execute (siempre) y submit

3	192.168.1.3	Submit
4	192.168.1.4	execute y submit

Tabla 3. 3 Asignación de direcciones a las máquinas de la red prototipo

Paso 4: ¿Donde deberá instalarse Condor?

Este paso decide donde se situarán los ficheros de Condor. En este caso en particular se seleccionó la instalación por defecto en c:\condor. La instalación en el disco local se decide por varias razones:

- Los servicios de Condor funcionan en el sistema local, y a través de Microsoft Windows, el sistema local no posee privilegios de red. Por lo tanto, para que funcione Condor, Condor debe instalarse en una partición de disco duro, a contrario de una partición de red (servidor de ficheros).
- La segunda razón para la instalación en un disco local es que el uso de Windows, posee implicaciones, dependiendo de donde se instale Condor.
- Es posible situar Condor en un disco local que no sea local, si esta dependencia es añadida a los servicios del control manager, por lo que Condor comienza después de que los servicios de ficheros requeridos sean correctos.

Paso 5: ¿Dónde se encuentra la máquina virtual de Java?

Es posible para Condor ejecutar trabajos en el Universo Java. Para poder tener soporte para java se debe facilitar un path para java.exe al sistema. Si se quiere deshabilitar el Universo de java, solo hay que dejar esa línea en blanco. En este Pool en particular se dejó el valor por defecto, para posibles posteriores usos del Universo de Java.

Figura 3. 4 Universo Java en Windows

Paso 6: ¿Dónde debe Condor mandar un e-mail si las cosas van mal?

Varias partes de Condor mandan correos al administrador de Condor, si algo va mal, y requiere atención humana. La dirección de e-mail y el SMTP de ese administrador.

Figura 3. 5 Servidor SMTP y dirección del administrador de Condor

Paso 7: El dominio

Este paso es eliminado en la instalación de un Condor personal. En el Pool de este proyecto se usará el dominio de la universidad.

Figura 3. 6 Dominio DNS o UID de la máquina

Paso 8: Permisos de acceso

Este paso se omite en la instalación de un Condor personal. Las máquinas del Pool de Condor, necesitarán varios tipos de permisos de acceso. Las tres categorías son:

Lectura: permite a una máquina obtener información acerca de Condor, así como del estado de las máquinas de Pool y las colas de trabajo. Todas las máquinas del Pool deben de tener acceso de lectura. Además, dar acceso de lectura a *.cs.wisc.edu permitirá al equipo de Condor obtener información sobre el Pool de Condor, si dicha información es necesaria.

Escritura: Todas las máquinas del Pool pueden tener permisos de escritura. Esto permite a las máquinas que se especifique mandar información a los demonios de Condor, por ejemplo, empezar un trabajo de Condor. Para que una máquina se una a un Pool de Condor, debe tener permisos de lectura y escritura por parte de todas las máquinas del Pool.

Administrador: Una máquina con acceso de administrador, poseerá permisos extendidos, para poder realizar cosas como: cambiar las prioridades de los usuarios, modificar la cola de trabajo, iniciar y parar los servicios, y reiniciar Condor. Se le deberá dar acceso de administrador al Central Manager, que es la opción por defecto. Este acceso estará garantizado para toda la maquina, por lo que habrá que tener cuidado.

Figura 3. 7 Permisos de administrador, lectura y escritura en Windows

En este caso, todas las máquinas del Pool, tendrán acceso de lectura y escritura, y cada máquina poseerá para sí misma permisos de Root.

Paso 9: Políticas de Trabajo

Condor ejecutará las tareas sometidas en las máquinas, en un orden de preferencia dado por la instalación. Se dan tres opciones, siendo la primera la más comúnmente usada por Condor. Esta especificación puede cambiarse, según las necesidades de las máquinas.Las tres *opciones* son:

- Después de 15 minutos de no haber actividad en la consola y baja actividad de la CPU.
- Ejecutar siempre tareas Condor.
- Después de 15 minutos de no haber actividad en la consola.

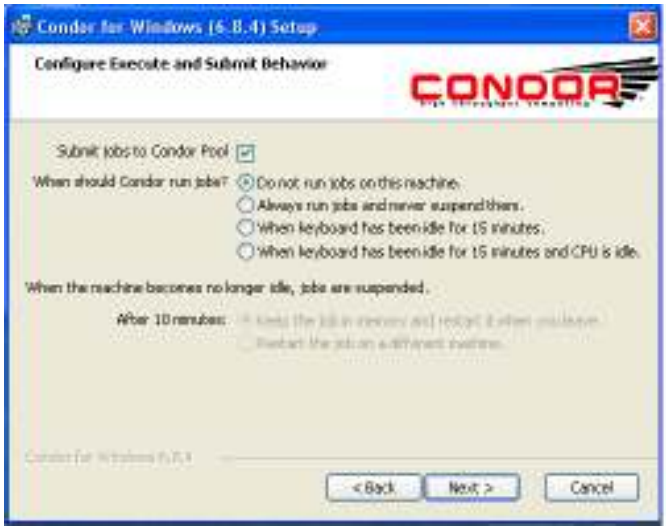


Figura 3. 8 Políticas de trabajo en Condor

Con actividad de la consola se refiere al uso del ratón o del teclado. Por ejemplo si uno está leyendo este documento online, y usa para ello tanto el ratón como el teclado para cambiar la posición del texto, se está generando actividad de consola.

Actividad baja de CPU se considera un porcentaje de carga inferior al 30% (lo cual puede ser modificado en el archivo *condor_config*). Si se posee una máquina con múltiples procesadores, este porcentaje se considera como la media de actividad de CPU de todos los procesadores. Si lo que se quiere es hacer una prueba general del funcionamiento de Condor, la mejor opción es la de ejecutar tareas siempre. La elección que se hizo para las máquinas de este caso fue:

Máquina	Papel	Política de Trabajo
1ª	Central Manager: Submit y Execute	Ejecutar tareas después de 15 minutos sin actividad de consola
2ª	Execute y Submit	Ejecutar tareas siempre
3ª	Submit	No ejecuta tareas de Condor (solo las lanza)
4ª	Execute y Submit	Ejecutar tareas siempre

Tabla 3. 4 Políticas de trabajo de las máquinas de la red prototipo

Paso 10: Política de eliminación de tareas

Este se paso se omite si se elige la opción (Paso 9) de que las tareas de Condor se ejecuten siempre en las máquinas. Si Condor está ejecutando una tarea, y el usuario vuelve a usar la máquina, Condor inmediatamente suspenderá el trabajo, y después de 5 minutos decidirá que hacer con el trabajo que se ha completado parcialmente. Hay dos opciones:

- *La tarea se elimina 5 minutos después de que el usuario regrese:* El trabajo se suspende inmediatamente cuando se detecta actividad de consola. Si la actividad de consola continúa durante 5 minutos, entonces el trabajo se elimina. Como en la versión para Windows no se incluye la

opción de check-pointing, la tarea deberá comenzar su ejecución desde el principio cuando se reinicie la ejecución. La tarea será puesta de nuevo en la cola.

- *Suspender el trabajo dejándolo en memoria:* La tarea se suspende inmediatamente. Después, cuando la actividad de la consola cese durante 10 minutos, la ejecución de la tarea de Condor será reanudada. El inconveniente de esta opción, es que en el momento que el trabajo debe permanecer en memoria, ocupará espacio de la misma. En determinadas ocasiones, sin embargo, este espacio de memoria en muy pequeño.

¿Qué opción elegir? Matar una tarea es menos intrusito, que dejarlo en memoria durante algún tiempo. Dejarlo en memoria, sin embargo, tiene el beneficio de que no se pierde el tiempo de ejecución que ya se ha utilizado para dicha tarea.

En el caso del Pool de este proyecto, como el objetivo es que muchas máquinas puedan ejecutar el servicio ofrecido por Condor, se ha optado por la primera opción, con el objetivo de no colapsar la memoria de las máquinas.

Paso 11: Revisión de la información introducida

Por último se recomienda hacer un repaso de los pasos anteriores, para asegurarse de que las opciones que se han seleccionado son las que interesan. Siempre se tiene la opción de novel hacia atrás en el cuadro de diálogo. Una vez finalizados estos pasos, antes de finalizar la instalación, se preguntará si se desea iniciar el servicio de Condor, a lo que se contestará que si. El directorio *c:\condor* ya ha sido creado así como los subdirectorios y archivos correspondientes.

3.2.4 Puesta en marcha de los demonios Condor.

Una vez finalizada la instalación de Condor, los demonios se inicializan de manera automática, por el propio programa de instalación. Si se quiere visualizar los demonios, puede hacerse mediante *ctrl.+alt+supr*, para visualizar en el administrador de sistema los demonios de Condor activos. En el caso de la *Figura 3.2.4.13* se puede observar los demonios master, schedd y startd por tratarse de una máquina Execute y Submit:

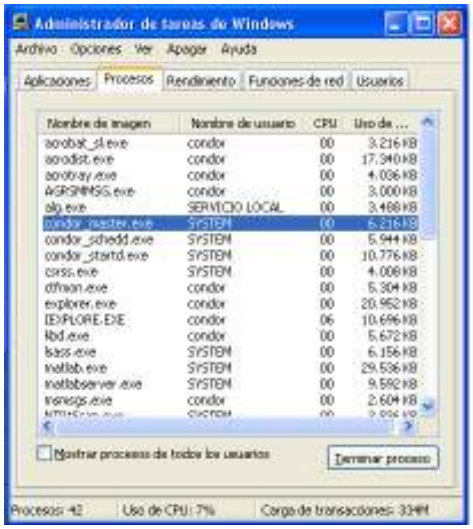


Figura 3. 9 Comprobación del funcionamiento de los demonios de Condor en Windows

Si la máquina en cuestión fuera el Central Manager deberían verse también los demonios negotiator y collector**.** Si por cualquier motivo no se hubieran iniciado los demonios no hubieran arrancado de manera automática, deberán arrancarse de manera manual. Hay tres maneras de arrancar el servicio de Condor

1. Reiniciando la máquina.
2. Panel de Control→Herramientas_Administrativas→Servicios→Condor→Start
3. En la línea de comandos, introduciendo *net start condor*

Una vez hecho esto el servicio de condor se iniciará de manera automática cada vez que se inicie la máquina.

Cuando el servicio de Condor se haya iniciado, lo siguiente que deberá hacerse es añadir nuestras credenciales, para poder empezar a lanzar tareas al Pool. Para hacerlo debe usarse el comando de Condor *condor_store_credd add*. Una vez introducido dicho comando se pedirá por pantalla que se introduzca un password. Ahí debe introducirse la clave de la propia cuenta de Windows, que en este caso como se indicó al inicio es condor. Si el password es correcto, Condor indicará que la operación se ha completado correctamente.

Inicialmente sólo pueden usarse los comandos de Condor desde la ruta *c:\condor\bin.* Para cambiar esto será necesario modificar la variable de entorno *PATH,* la cual que encuentra en Mi PC→Propiedades del Sistema→Opciones Avanzadas→Variables de entorno→PATH→Modificar. Sólo se necesita añadir a las rutas que hay, la ruta *c:\condor\bin* y ya se podrán usar los comandos de Condor desde cualquier lugar.

Figura 3. 10 Modificación de la variable PATH

3.3 Condor Pool en S.O Linux.

3.3.1 Entorno del Condor Pool

El entorno del Pool de Condor es el mismo que el que se indicó en el apartado 3.2.1 de la instalación del Condor en Windows.

3.3.2 Requisitos Hardware

Las características del hardware, son exactamente las mismas que las que se describieron en el apartado 3.2.2.

3.3.3 Requisitos Software

Antes de comenzar la instalación del Pool, es necesario llevar a cabo como Root, la instalación de NIS y NFS en las máquinas que forman parte del Pool.

3.3.3.1 Instalación de NIS

Para ejecutar el software de servidor/cliente de NIS se necesitará ejecutar antes el programa *portmap*. En RedHat Linux existe un script en */etc/init.d/portmap* para arrancar éste demonio. Todo lo que tienes que hacer es añadirlo a un runlevel si deseas que se realice en el arranque.

El mapeador RPC es un servidor que convierte números de programas RPC en números de puerto de protocolo TCP/IP (o UDP/IP). Debe estar ejecutándose para poder realizar llamadas RPC (que es lo que el software de cliente NIS hace) a servidores RPC (como un servidor NIS) de esa máquina. Cuando un servidor RPC arranca, avisará al mapeador de puertos por cuál puerto está escuchando, y a qué números de programas RPC está preparado para servir. Cuando un cliente desea hacer una llamada RPC a un número de programa dado, primero deberá contactar con el mapeador de puertos de la máquina servidora para determinar el número de puerto al que los paquetes RPC deben ser enviados. El comando *rpcinfo* nos dará la información de que programas está preparado para servir.

El software necesario para poner en marcha y configurar un servidor o cliente de NIS en una distribución RedHat Linux, es el siguiente:

ypserv

yp-tools

ypbind

Instalación del Cliente NIS

Hay que asegurarse de que están instalados los paquetes ypbind e yp-tools. El programa /usr/sbin/ypbind es el servicio cliente de NIS, que nos permitirá interactuar con un Servidor de NIS y acceder a los mapas que este sirva (/etc/password, /etc/group). Para comprobar que ypbind funciona correctamente, habría que seguir los siguientes pasos:

1. Tener arrancado el portmap para lo que se usará el comando rpcinfo –p

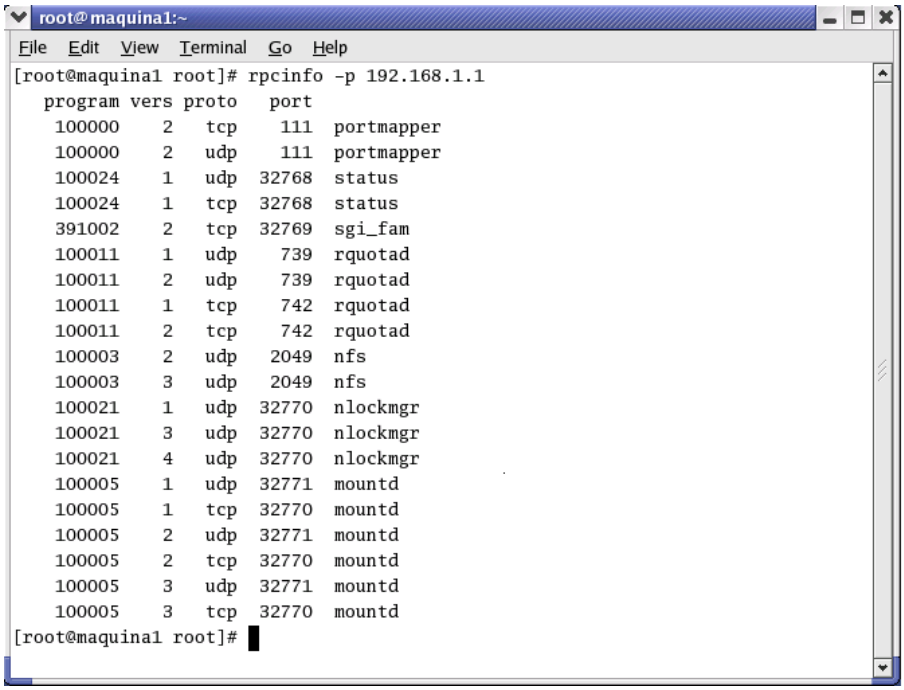


Figura 3. 11 Verificación del funcionamiento del portmap

2. Definir el dominio de NIS al cual sirve el servidor con el comando *domainname*. En nuestro caso el dominio es *upct.es*:

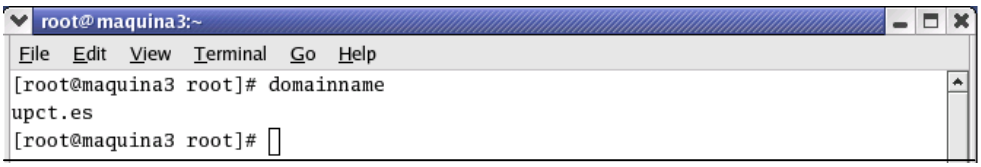


Figura 3. 12 Comprobación del dominio NIS de la máquina

Figura 3.3.3.1. 1

3. Crear el directorio /var/yp si no existe.
4. Iniciar el demonio ypbind /usr/sbinypbind.
5. Usar el comando rpcinfo –p localhost para comprobar si ypbindes capaz de registrar su servicio con el maleador de puertos.

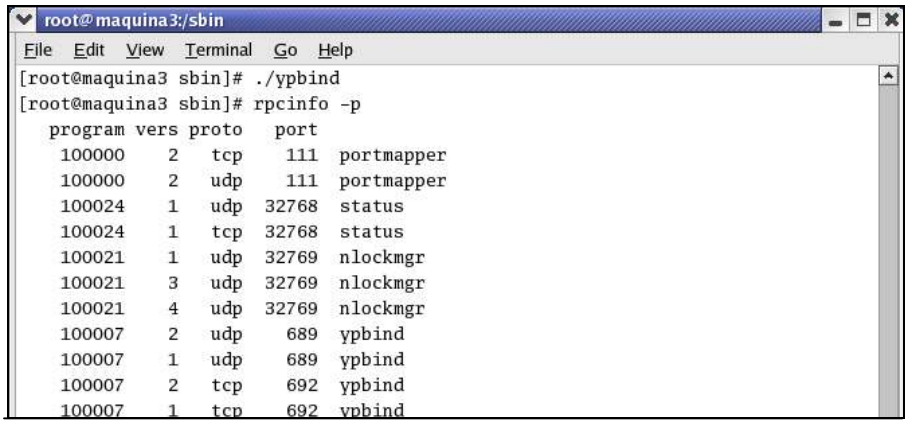


Figura 3. 13 Comprobación del funcionamiento del ypbind

6. Comprobar que puedes acceder a los mapas que sirve el servidor NIS con el comando ypcat passwd. Esto debería devolver el mapa passwd del servidor.

Si todo ha funcionado correctamente, debe automatizarse este proceso para que se realice cada vez que arranque el cliente. Para ello hay que modificar el script */etc/rc.d/init.d/ypbind* que acepta {start|stop} para lanzar o parar el servicio.

Instalación del Servidor NIS

Habrá que comprobar que está instalado el paquete ypserv (*rpm -q ypserv*), que es el que nos permitirá instalar un servidor de páginas amarillas o NIS. Habrá que editar el fichero */etc/ypserv.conf* que nos permitirá configurar algunas opciones para el sevidor ypserv. Normalmente contiene una lista de reglas para hosts especiales y reglas de acceso a los mapas. El fichero */var/yp/securenets* define los derechos de acceso al servidor NIS por parte de los clientes. El fichero contiene pares netmask/network de forma que si la dirección IP del cliente se corresponde con la entrada podrá actuar como cliente de NIS.

Hay que asegurarse que el portmap está corriendo (*rpcinfo -p localhost*) y entonces y solo entonces lanzar el demonio ypserv. El siguiente paso sería generar las bases de datos (mapas) de NIS. En el servidor maestro habrá que ejecutar */usr/lib/yp/ypinit –m*

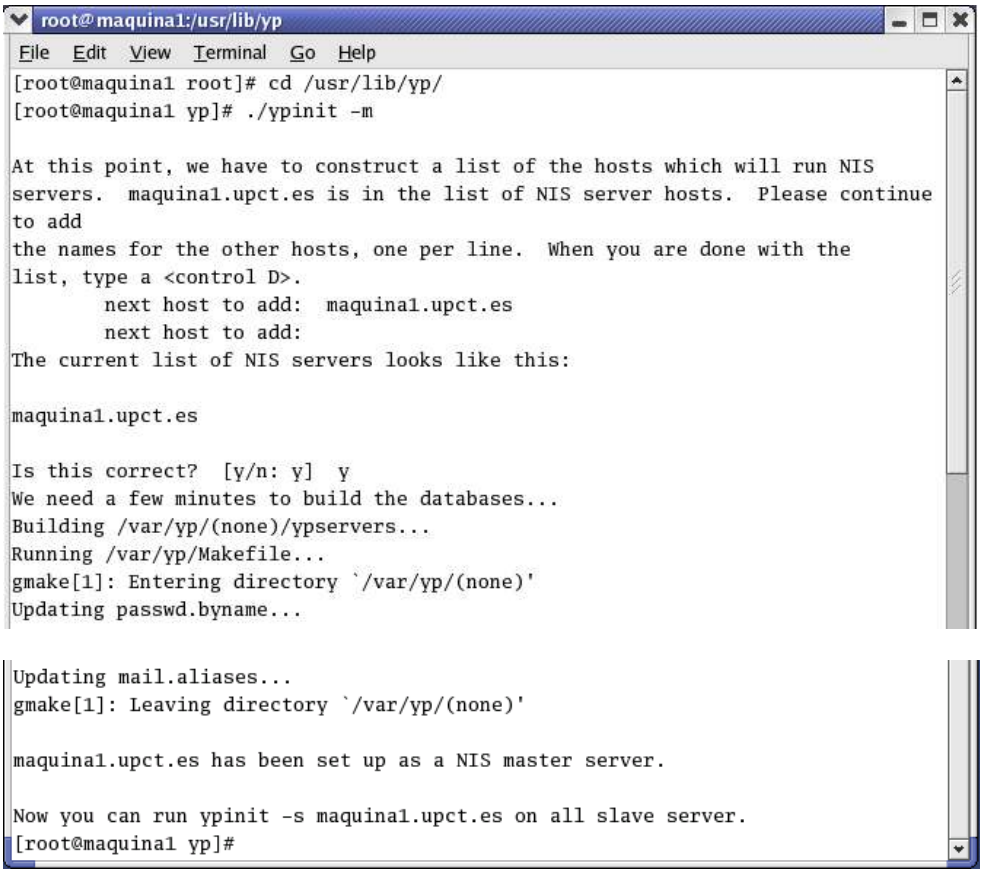


Figura 3. 14 Creación de los mapas de NIS

Por último hay que ejecutar el demonio ypserv , cuyo funcionamiento podrá comprobarse ejecutando rpcinfo –p localhost, debiendo aparecer algo parecido a:

100004 2	udp	965	ypserv
100004 1	udp	965	ypserv
100004 2	tcp	968	ypserv
100004 1	tcp	968	ypserv

3.3.3.2 Instalación de NFS

El *Network File System* (Sistema de archivos de red), o NFS, es un protocolo de nivel de aplicación, según el Modelo OSI. Es utilizado para sistemas de archivos distribuido en un entorno de red de computadoras de área local. Posibilita que distintos sistemas conectados a una misma red accedan a ficheros remotos como si se trataran de locales. Para la instalación necesitaremos tener instalado portmap y el paquete nfs (nfs-utils) que se puede encontrar en la mayoría de las distribuciones en el ordenador que vaya a hacer de servidor de disco. El portmap nos permitirá realizar conexiones RPC al servidor y es el encargado de permitir o no el acceso al servidor a los equipos que especifiquemos. Para saber si se tiene el portmap instalado bastará con un simple

>> *ps aux | grep portmap*

Para saber si NFS está en marcha haremos una consulta al portmap para que nos indique qué servicios tiene en marcha

```
>> rpcinfo -p
```

Cuya salida debería ser algo similar a:

```
100003  2       udp     2049    nfs
100003  3       udp     2049    nfs
```

Para realizar la instalación en el servidor debe hacerse lo siguiente:

1) Botón de inicio → Server Settings → NFS Server
2) Add
3) Se rellena el formulario para agregar el directorio */home* que contendrá a los usuarios, con permisos de *lectura y escritura* y para cualquier ordenador de la red.

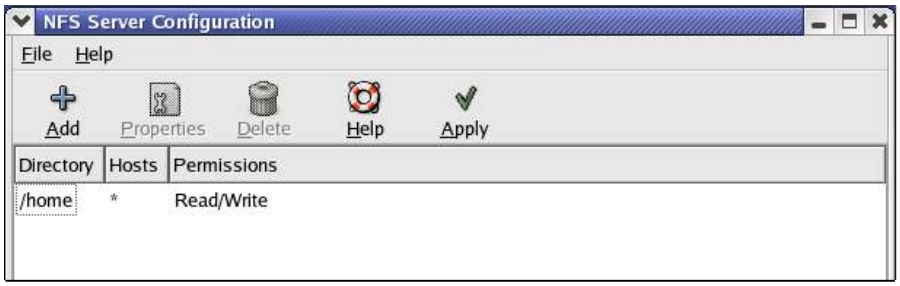


Figura 3. 15 Creación del demonio NFS

En los clientes será necesario editar el directorio /etc/exports, para indicar que se desea compatir, quien es el servidor, y con que permisos se va a compartir:

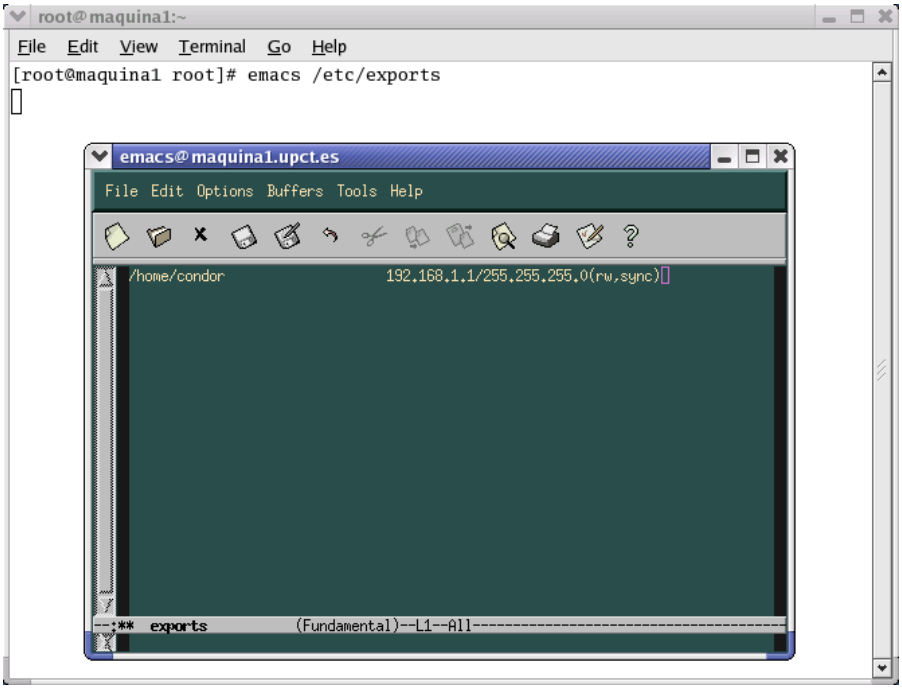


Figura 3. 16 Modificación del fichero exports

En el caso de este proyecto en particular, se decidió compartir el directorio /home/condor, tomando como servidor NFS la Máquina 1, dando permisos de lectura y escritura a los clientes. Otra manera de comprobar el correcto funcionamiento de NFS sería usando la combinación de menús:

1) Botón de inicio → Server Settings → Services
2) Se inicia si fuera necesario.

Figura 3. 17 Activación del demonio NFS

3.3.4 Instalación del software Condor

Una vez se ha instalado NIS y NFS, el primer paso en la instalación será el crearse una cuenta Condor. Para crear una cuenta de usuario nueva en Redhat, ejecutar Menú de Inicio→System Settings→Users and Groups→Add User. En este caso se ha optado por una cuenta con el nombre de condor, la clave condor, y directorio, local en */home/condor*. De este modo ya se está preparado para proceder a la instalación de condor.

El primer paso para realizar la instalación es descargar Condor de la página oficial: http://www.cs.wisc.edu/condor/downloads. Para poder proceder a la descarga, primero hay que registrarse como ya se explicó en el punto 3.2.4. En este caso se bajó *condor-6.8.5-linux-x86-redhat80.tar.gz* que era el más adecuado a la arquitectura de las máquinas como ya se indicó anteriormente.

Después de la descarga, primero deben descomprimirse los archivos. Para ello es necesario ejecutar en el directorio */root*:

tar -xzf linux-x86-redhat80.*tar.gz*

Una vez descomprimido podrán observarse los siguientes archivos:

- DOC Direcciones donde se puede encontrar información sobre Condor.
- INSTALL Direcciones de instalación.
- LICENSE.TXT La licencia de Condor
- README Información general de Condor.
- *condor_install* El script Perl usado para instalar y configurar Condor.
- examples/ Directorio que contiene ejemplos de programas en C,C++ y Fortran, para ejecutar en Condor.
- *release.tar* fichero que contiene los binarios y las librerías de condor

Gracias a esto se crearán los subdirectorios */bin, /etc, /include, /sbin, /lib,* como se observa en la siguiente figura:

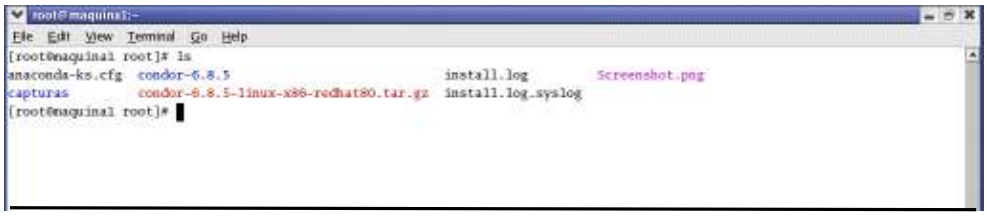


Figura 3. 18 Descomprimir el archivo de instalación

Hay dos maneras de llevar a cabo la instalación: de manera simplificada o paso a paso.

- *De manera simplificada*

Habrá que tener en cuenta los siguientes argumentos de entrada:

1. *--install=/path/to/release.tar*. Ruta donde se encuentra el archivo release.tar. En este caso */root*
2. *–install-dir=directory.* Ruta donde se quiere instalar el directorio condor. En este caso */usr/local/condor*
3. *–local-dir==directory.* Ruta del directorio local. En este caso */home/condor*
4. *–type=manager,execute,submit.* Esta opción los distintos papeles que la máquina desempeñará en el Pool de condor: Central Manager, Submit o Execute. Los distintos papales que la máquina desempeñe irán separados por comas.

Hay que configurar condor primero en la máquina que funcione como Central Manager. Cuando se tenga en claro como se quiere configurar la máquina en cuestión, habrá que ejecutar *condor_configure* seguido de las distintas opciones que se deseen.

- *Paso a paso*

Esta fue la opción por la que se optó en la instalación del Pool. Para iniciar la instalación se ejecutará desde el directorio */root/condor-6.8.4/* el comando:

perl condor_install

Una vez hecho esto comenzará una instalación guiada, que queda descrita en los siguientes pasos.

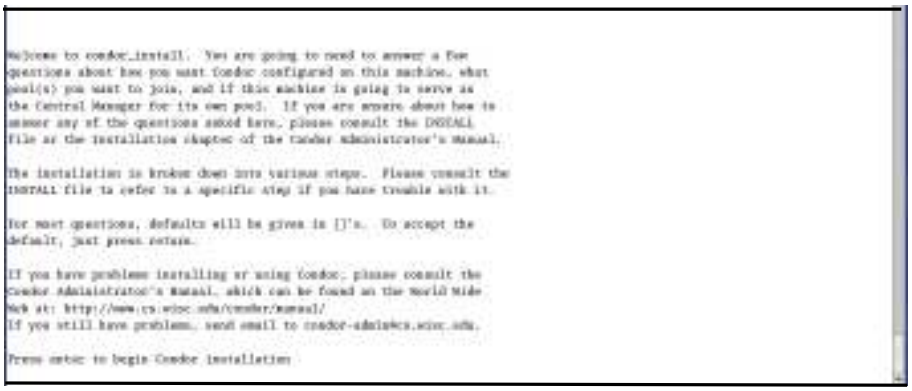


Figura 3. 19 Inicio de la instalación mediante condor_install

Paso1: ¿Qué tipo de instalación de condor se desea?

Hay tres tipos de instalación para elegir: ‘submit-only’,’full-install’, y ‘central manager‘. Una máquina Submit Only puede enviar tareas al Pool, pero no podrán

ejecutarse tareas de condor en dicha máquina. Una máquina Full-Install puede lanzar y ejecutar tareas.

Si se quiere ejecutar tareas de Condor, se debe instalar y ejecutar Condor como Root o como usuario condor.

Si lo que se desea es instalar una máquina del tipo Submit Only, se puede instalar Condor como Root o como usuario condor, o se puede instalar Condor por sí mismo en el directorio */home*.

La otra posible instalación es la creación de una máquina que actúe como Central Manager. Si se hace una instalación del tipo *full-install*, y se indica que se desea que el local host sea el Central Manager, este paso lo hará de manera automática. Sólo se debe elegir la opción *central manager* en el paso 1 si se ha ejecutado "condor_install" en el servidor de ficheros, y ahora se quiere ejecutar "condor_install" en una máquina diferente, que debe ser el Central Manager. En este caso se ha optado por una instalación del tipo:

Máquina	Papel	Tipo de instalación
1ª	Central Manager, Submit, Execute	full-install
2ª	Submit, Execute	full-install
3ª	Submit, Execute	full-install
4ª	Submit	submit-only

Tabla 3. 5 Tipo de instalación de las máquinas de la red prototipo

Paso2: ¿Cuántas máquinas se quieren establecer con este tipo de instalación?

Si se desea instalar Condor en múltiples máquinas, y se posee un sistema de archivos compartido, entonces Condor preguntará por el hostname de cada una de las máquinas que se desea añadir al Pool. Si no se posee un sistema de archivos compartido, se debe ejecutar condor_install en cada una de las máquinas donde se desea instalar Condor, por lo que Condor no preguntará por los nombres de las máquinas. Si se le provee de dicha lista, el programa de instalación creará en todas las máquinas los directorios y ficheros necesarios. Al final Condor volcará esta lista a un fichero *roster* el cual puede ser usado por los scripts para ayudar a mantener el Pool.

En este caso, se desea que se puedan añadir máquinas al Pool de manera dinámica, por lo que, a pesar de poseer un sistema de ficheros compartido, no se puede dar una lista de las máquinas fijas que usarán el Pool. Por ello debe de contestarse que no a esa pregunta.

Paso 3: Instalar el directorio Condor /release

El directorio */release* contiene 5 subdirectorios: */bin, /etc, /lib, /libexec* y */sbin.*

- */bin* contiene programas ejecutables a nivel de usuario
- */etc* es la localización recomendada para los ficheros de configuración de Condor, y a su vez posee un subdirectorio */examples*, con los ficheros de configuración por defecto, y otros ficheros empleados para la instalación de Condor.

- */lib* contiene librerías para enlazar los programas de usuario Condor, y los scripts usados por Condor.
- */sbin* contiene todos los programas administrativos ejecutables, y los demonios de Condor.
- */libexec* contiene programas que sólo Condor necesita ejecutar.

Si se poseen varias máquinas con un sistema de archivos compartido, que deben ejecutar Condor, hay que poner el directorio *release* en dicho sistema de ficheros, de modo que sólo se posea una copia de todos los binarios, de modo que cuando sea necesario actualizarlos, sólo será necesario hacerlo en un lugar. Hay que tener en cuenta que el directorio release depende de la arquitectura, por lo que a la hora de descargarse los binarios, hay que tener en cuenta la arquitectura de cada una de las máquinas del Pool.

condor_install intentará de encontrar algún directorio release instalado anteriormente:

- Si no puede encontrar ninguno, preguntará si se ha instalado alguno anteriormente. Si no se ha instalado ninguno, el programa intentará instalarlo por sí mismo descomprimiendo el directorio release (que quizá no se descomprimió anteriormente)
- Si ya se ha instalado el directorio release, preguntará si la ruta donde se encuentra instalado, es el que se desea para la instalación. Si no es así debe indicarse la ruta que se desea.

Se ha optado en este caso por la ruta de instalación */usr/local/condor* (que es la ruta por defecto), para que el programa descomprima el release.tar.

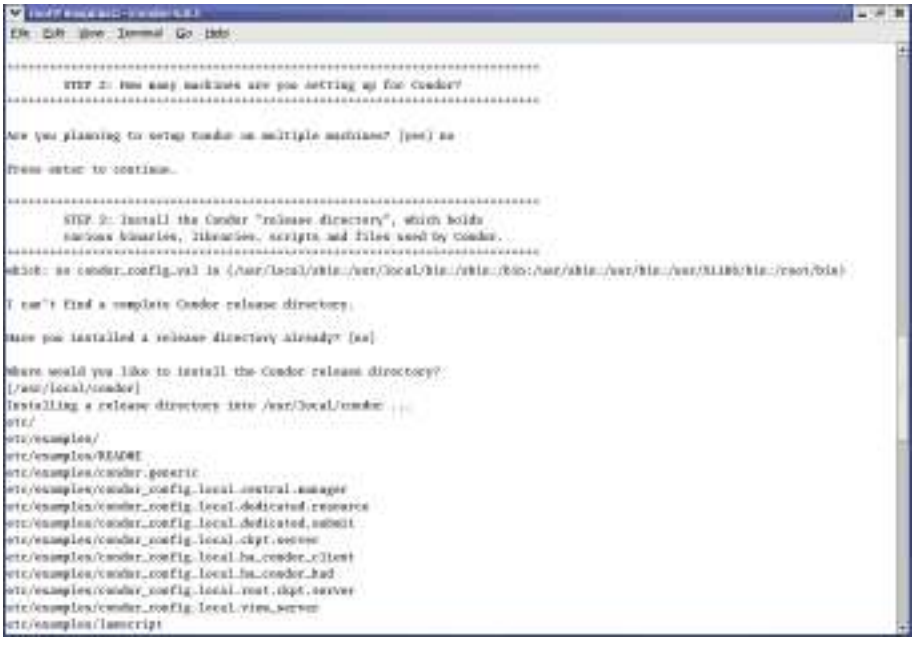


Figura 3. 20 Descomprimir el archivo release.tar

Si se desea descomprimir el archivo release.tar de manera manual, sólo habrá que ejecutar en */root/condor-6.8.5:*

tar –xvf release.tar.

Si lo que desea es creación de un Central Manager (se debe haber elegido la opción Central Manager en el paso 1), el paso número 3 es la última pregunta que se necesita responder.

Paso 4: ¿Dónde debe Condor mandar un e-mail si algo va mal?

Varias partes de Condor, mandarán un correo al administrador de Condor si algo va mal y necesita de atención humana para solucionarlo. Para ello se necesita especificar la dirección e-mail del administrador.

También se necesita especificar la ruta completa para el programa “mail” que Condor usa para enviar el e-mail. El programa mail necesita comprender la opción –s que Condor emplea para enviar el e-mail. La opcion por defecto que aparece será probablemente la correcta. En las máquinas Linux, como hay muchas distribuciones e instalaciones distintas, hay que verificar cual es la que usa por defecto. Si el programa de instalación parece no encontrar el programa mail que está especificado, habrá que buscar la ruta del programa con el comando:

wich mail

Si no se consigue encontrar la ruta se puede probar con:

wich mailx

Si aun así no se puede encontrar, deberá consultarse al administrador del sistema. Para verificar si admite la opción –s se puede consultar la página man.

En este Pool se mandarán correos al Central Manager (que actúa como administrador del sistema), estando el programa mail en */bin/mail* (ruta por defecto).

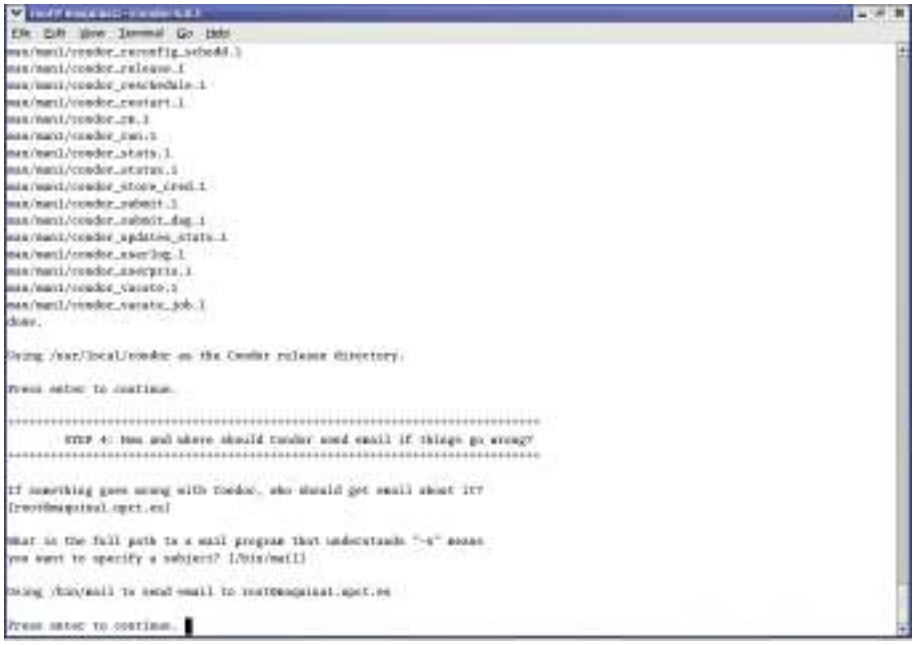


Figura 3. 21 Dirección del administrador en Redhat

Paso 5: Sistema de ficheros y dominios UID

Condor no tiene dependencia de un sistema de ficheros y un espacio UID común fuera del Universo Standard. Los trabajos pueden especificar el fichero de transferencia en al archivo submit. Para tareas del Universo Standard si será necesario. Para usar las ventajas de un sistema de archivos (como por ejemplo NFS), se necesita un UID común. Es importante configurar condor de manera correcta en un sistema de ficheros distribuido. Se preguntará si se posee un sistema de ficheros distribuido. Si es así, condor_install configurará el FILESYSTEM_DOMAIN para ser el dominio de la máquina donde se está ejecutando *condor_install*. Si no es así se deberá poner:

FILESYSTEM_DOMAIN=$(FULLHOSTNAME)

indicando que cada máquina posee su propio dominio. Para el dominio UID Condor debe saber si todas las máquinas que forman parte del Pool poseen el mismo UID. Si es así, UID debe ser establecido como el dominio propio de cada máquina. Si no es así se establecerá como:

UID_DOMAIN=$(FULLHOSTNAME)

indicando que cada máquina posee su propio dominio. Si se posee un UID común, *condor_install* preguntará si se trata de un dominio UID "soft", significando, que aunque se tenga un único UID, no todas las máquinas en el Pool, tienen a todos los usuarios en ficheros password individuales.

En la solución dada para este Pool en concreto, se ha optado por un sistemas de ficheros compartido en el dominio upct.es, un UID único (upct.es), por si se quisieran ejecutar tareas dentro del Universo Standard, y todos los usuarios tendrán ficheros password individuales.

Paso 6: Soporte al Universo Java en Condor

Condor tiene la posibilidad de ejecutar programas de Java con entrada y salida remota, pero sin checkpointing. Si se quiere habilitar esta posibilidad en Condor, habrá que indicar que si se quiere establecer el Universo Java en la máquina. El instalador intentará establecer si se posee una JVM (Java Virtual Machine) válida y si no encuentra ninguna se lo indicará al usuario. Si no se desea ejecutar el Universo Java (como es este caso), entonces es más seguro contestar que no en esta parte.

Paso 7: ¿Donde deben instalarse los programas públicos?

Es recomendable que los programas de Condor (a nivel de usuario se instalen en el directorio release). De esta manera, cuando se quiera instalar una nueva versión de los binarios de Condor, sólo será necesario reemplazar el directorio release y todo será actualizado. Por eso una opción es tener a los usuarios condor en *<release_dir>/bin*. Sin embargo la opción que se recomienda, y que se ha empleado para este Pool, es crear enlaces simbólicos al directorio */usr/local/bin*, y que desde allí se llame a los programas de Condor. *condor_install* hace esto si se desea. Lo único que debe hacerse es indicarle hacia que directorio quieres hacer el enlace. De esta manera los usuarios no tienen que cambiar su path para usar Condor, y se pueden tener los binarios instalados en su propia localización.

Si no se está instalando Condor ni como Root, ni como usuario Condor hay un script perl, que funciona de envoltorio a todas las herramientas de Condor que se crean, establece las variables de entorno adecuadas, y automáticamente le pasa ciertas opciones a las herramientas. Todo esto se hace de manera automática por condor_install, para lo cual se necesita indicarle a Condor donde pone el script perl. El script por sí mismo. Este paso puede observarse en la Figura 3.3.4.7. A partir de es donde se diferencian las instalaciones:

❖ Instalación Full Install

Paso 8: ¿Qué máquina va a ser el Central Manager?

Hay que escribir el hostname completo de la máquina que se ha seleccionado para que sea el Central Manager. Si *condor_install* no puede encontrar información sobre el host que se le ha indicado, mostrará un mensaje de error y te pedirá que se solucione el problema.

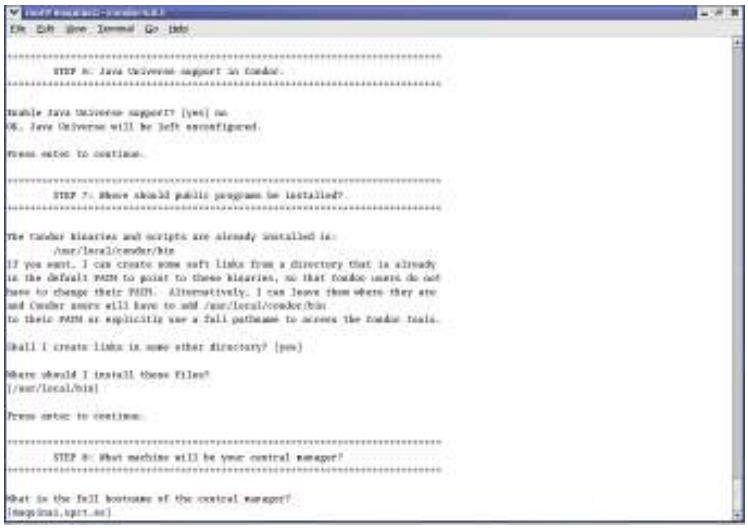


Figura 3. 22 Instalación de los programas públicos y elección del Central Manager

En el Pool de este proyecto el Central Manager es la máquina 192.168.1.1 (maquina1.upct.es), que es el hostname que se le pasa en este paso.

Paso 9: ¿Dónde debe situarse el directorio local?

Este es el directorio que se comentó en el quinto punto del apartado 3.2.4. *condor_install* trata de hacer una serie de conjeturas sobre el directorio que se desea usar para dicho propósito. Se puede contestar que sí cuanto te pregunte por el directorio que se desea, o bien escribirlo. Como el directorio debe de ser único, es normal usar el hostname de cada máquina. Cuando se está escribiendo en el propio path, se puede emplear '$(HOSTNAME)', que Condor tomará como el hostname de la máquina en la que se ejecuta. A partir de este momento Condor intentará crear los directorios necesarios en todas las máquinas que se indicaron en el Paso 2.

Una vez que se haya seleccionado el directorio local, *condor_install* creará los subdirectorios necesarios con los permisos adecuados. Deben de tener los siguientes permisos y propiedades:

```
dwxr-xr-x    2 condor   root 1024 Sep     6 15:35 execute/
dwxr-xr-x      2 condor   root       1024 Sep     6 15:35 log/
dwxr-xr-x      2 condor   root       1024 Sep     6 15:35 sPool/
```

En este proyecto el directorio local se ha situado en */usr/local/condor/home.*Si el directorio local se encuentra en un sistema de archivos compartido Condor te preguntará por la localización de los archivos lock. En este caso, cuando condor_install termina, hay que ejecutar condor_init en cada máquina del Pool para crear el directorio lock antes de lanzar Condor.

Paso 10: ¿Dónde deben situarse los ficheros de configuración local?

Hay diferentes niveles dentro de los ficheros de configuración. Hay un fichero de configuración local que debe instalarse en */usr/local/condor/etc/condor_config*, y hay otros que son específicos de la máquina, que pueden sobrescribir características del fichero de configuración global. Si se ha instalado Condor en múltiples máquinas o se está configurando la máquina que actúa como Central Manager, se debe elegir la ruta de instalación de los ficheros de configuración locales. as dos opciones son:

- Tener un solo directorio que posea todos los ficheros de configuración local que se llame $(HOSTNAME).local. Esto se recomienda cuando se tiene un sistema de archivos distribuido, ya que es más sencillo configurar un Pool desde un ligar centralizado

- Tener ficheros de configuración independientes en los directorios locales de cada máquina

Debido a que no se puede establecer con exactitud el número de máquinas que van a formar parte del Pool (nuevas máquinas se incorporan para lanzar tareas, se añaden nuevas máquinas en los laboratorios), se ha optado por la segunda opción, situándose el fichero de configuración local en /usr/local/condor/etc/$(HOSTNAME).local. Cuando se ha hecho este paso, el programa de instalación, le indica al archivo de configuración global la situación del fichero de configuración local. demás si se trata de la máquina que actúa como Central Manager (como es el caso de la 3.3.4.9) éste será el momento en el que se active la máquina como Central Manager.

Una vez hecho esto, se pedirá (en el caso de que sea el Central Manager) el nombre del Pool. En este caso en particular se optó por **PROYECTO-UPCT-CONDOR.**

Paso 11: ¿Dónde debe Condor buscar estos ficheros de configuración?

Hay una serie de lugares conocidos, donde Condor busca los ficheros de configuración, por lo que es recomendable hacer enlaces desde alguno de ellos hacia */usr/local/condor/ect/condor_config*. De esa manera se puede tener guarda la configuración de Condor en un lugar centralizado, pero todos los demonios de Condor serán capaces de encontrar sus ficheros de configuración. También se le puede dar a la variable de entorno CONDOR_CONFIG el valor */usr/local/condor/etc/condor_config*.

Una vez que se completa este paso de la instalación full_install, condor_install muestra mensajes describiendo que hay que hacer una vez que se termina la instalación. n el Pool del proyecto se ha elegido este tipo de instalación para la mayor parte de las máquinas. La idea es que los ordenadores que vayan a ejecutar tareas (es decir, los que formen parte del laboratorio) tengan este tipo de instalación, con un Central Manager por laboratorio.

❖ Instalación Submit Only

Esta instalación implica que la máquina podrá lanzar tareas de Condor en uno o más Pools. Para este tipo de instalación es necesario que la máquina se subscriba a dichos Pools. ra instalarlo hay que volver al paso 6 y continuar.

Paso 6: continuación

Se tiene la opción de lanzar tareas a más de un Pool. Para poder inscribirse a un Pool, es necesario indicar el hostname de cada máquina del Pool. El primer Pool que se indique será el que se tome por defecto para comenzar y lanzar tareas. ay un fichero de configuración para cada Pool. La localización de cada fichero debe especificarse.

La identificación de cada Pool requiere un único nombre. Un último apartado relaciona cada Pool con un nombre. El nombre debe ser el argumento para la opción **–Pool** de la línea de comandos.

En el Pool del proyecto, sólo se ha llevado a cabo esta instalación el la máquina 4ª, pero la idea es que los usuarios externos lleven a cabo esta instalación, de modo que sólo tengan que lanzar tareas al Pool

Una vez que Condor ha sido instalado en las máquinas que van a formar parte del Pool, hay algunas cosas que deben comprobarse antes de realizar la instalación.

1) Hay que mirar el fichero de configuración global (/usr/local/condor/etc/condor_config). Hay algunas características a las

que se debe echar un vistazo, para asegurarse de que el Pool funciona como debe.

2) Condor puede monitorizar la actividad del teclado y la del ratón. Esto puede indicarse, modificando la variable CONSOLE_DEVICES de la sección condor_startd del archivo de configuración. Esta variable es una lista separada por comas, por lo que se puede tener cualquier dispositivo en el directorio */dev* como 'console devices' cuya actividad le será mandada al demonio **condor_startd** como *ConsoleIdleTime*.

3) (Sólo para Linux). Condor necesita ser capaz de encontrar el fichero utmp. Este fichero debe encontrarse en el directorio /var/run/utmp. Si no lo encuentra en dicho directorio, lo buscará en /var/adm/utmp. Si aún así no lo encuentra se da por vencido. Por ello si está en otro lugar se deben crear enlaces simbólicos desde donde se sitúen hacia */var/run/utmp*.

Lo siguiente que se debe hacer es ejecutar **condor_init** en todas las máquinas que van a formar parte del Pool para crear los directorios LOCK , ya que en este caso no han sido creados por el programa condor_install. Además con el comando condor_init se asegura que han sido creados todos los directorios que serán imprescindibles en Condor (/log, /sPool, /execute) así como también el fichero de configuración local, y el fichero de configuración global. Para ello es necesario situarse en */usr/local/condor/sbin/* y ejecutar **./condor_init**

Figura 3. 23 Fin de la instalación

Además de esto si se instala Condor en un sistema de ficheros distribuido, condor_init crea enlaces hacia cada una de las máquinas que van a formar parte del Pool y se indican en el fichero de configuración. Estos enlaces se hacen para que las máquinas puedan encontrar el fichero de configuración global, pero este no el caso de esta instalación. ambién debe copiarse el script *condor_generic* de la ruta */usr/local/etc/examples/condor_generic* a */usr/local/bin/condor.* Una vez copiado, deben de realizarse los siguientes cambios:

```
#!/usr/bin/env perl


$default_Pool="default";
$release_dir="/usr/local/condor";
$sbin="$release_dir/sbin";
$bin="$release_dir/bin";
%configlocation = (
                "default","/usr/local/condor/etc/condor_config"
);
```

Para que Condor se inicie cada vez que arranque la máquina, se debe copiar el script *condor_boot,* localizado en la ruta */usr/local/condor/etc/examples/condor.boot* en la siguiente ruta

[condor@maquina1 examples]$*cp –p /uss/local/condor/etc/examples/condor.boot /etc/init.d/condor*

Se debe crear un enlace simbólico entre los ficheros que se citan a continuación:

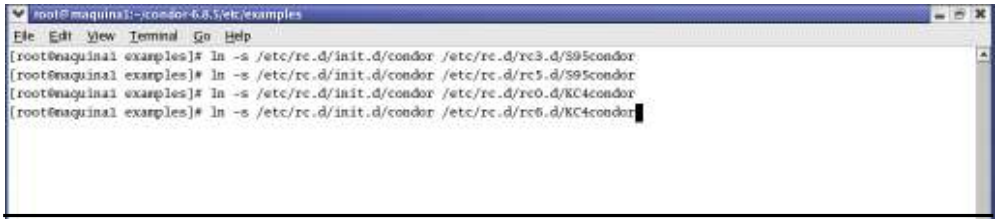


Figura 3. 24 Enlaces simbólicos para el inicio de Condor

El comando *ln* crea un enlace entre un fichero existente y un nombre nuevo de fichero. Ambos son idénticos para el usuario. La opción *–s* realiza un enlace simbólico a pesar del enlace existente. ara una completa instalación del script **condor_compile** se debe cambiar el nombre el enlazador ld de la ruta */usr/bin* por ld.real, y copiar el enlazador de Condor en dicha ruta:

[condor@maquina1 bin]$ *mv /usr/bin/ld /usr/bin/ld.real*

[condor@maquina1 bin]$ *cp –p /usr/local/condor/lib/ld /usr/bin/ld*.

El siguiente paso, como ya se indicó en el punto 1 es la modificación de los archivos de configuración:

- **Fichero de configuración global**

Las modificaciones realizadas en el archivo de configuración global han sido señalizadas, así como aquellos parámetros que se consideran interesantes. Gan parte de los parámetros, han sido establecidos por defecto gracias al script condor_install.

Se han eliminado aquellas partes que no son importantes en una instalación básica, y aquéllas que serán utilizadas para funciones específicas más adelante.

En este Pool en específico fue fundamental el cambio del parámetro *RunBenchmarks* a False, ya que debido a las características especiales de las máquinas del laboratorio, este parámetro no funcionaba bien, e impedía que las máquinas pasaran del estado *Owner* al de *Unclaimed* y por lo tanto hacía imposible que las máquinas pudieran ejecutar tareas.

[condor@maquina1 etc]$ /usr/local/condor/etc/ emacs condor_config

```
##  Part 1:  Settings you must customize:
##########################################################################
##########################################################################
LOCAL_DIR          = $(RELEASE_DIR)/home
#LOCAL_DIR         = $(RELEASE_DIR)/hosts/$(HOSTNAME)

#LOCAL_CONFIG_FILE      = $(LOCAL_DIR)/condor_config.local
LOCAL_CONFIG_FILE = $(RELEASE_DIR)/etc/$(HOSTNAME).local

UID_DOMAIN  = upct.es

FILESYSTEM_DOMAIN = upct.es
```

```
COLLECTOR_NAME          = PROYECTO-UPCT-CONDOR

##--------------------------------------------------------------------
##  Part 2:  Settings you may want to customize:
##  (it is generally safe to leave these untouched)
##--------------------------------------------------------------------
##--------------------------------------------------------------------

CONDOR_IDS=0.0

HOSTALLOW_ADMINISTRATOR = $(CONDOR_HOST)

HOSTALLOW_OWNER = $(FULL_HOSTNAME), $(HOSTALLOW_ADMINISTRATOR)

HOSTALLOW_READ =192.168.1.*

HOSTALLOW_WRITE = 192.168.1.*

HOSTALLOW_NEGOTIATOR = $(CONDOR_HOST)

HOSTALLOW_WRITE_COLLECTOR = $(HOSTALLOW_WRITE), $(FLOCK_FROM)
HOSTALLOW_WRITE_STARTD    = $(HOSTALLOW_WRITE), $(FLOCK_FROM)
HOSTALLOW_READ_COLLECTOR  = $(HOSTALLOW_READ), $(FLOCK_FROM)
HOSTALLOW_READ_STARTD     = $(HOSTALLOW_READ), $(FLOCK_FROM)

USE_NFS           = True


USE_AFS           = False

MAX_JOBS_RUNNING  = 200

LOCK        = /var/lock/condor

CREATE_CORE_FILES = False

##--------------------------------------------------------------------
##  Part 3:  Settings control the policy for running, stopping, and
##  periodically checkpointing condor jobs:
##--------------------------------------------------------------------

MINUTE            = 60
HOUR        = (60 * $(MINUTE))
StateTimer  = (CurrentTime - EnteredCurrentState)
ActivityTimer     = (CurrentTime - EnteredCurrentActivity)
ActivationTimer = (CurrentTime - JobStart)
LastCkpt    = (CurrentTime - LastPeriodicCheckpoint)

STANDARD    = 1
PVM         = 4
VANILLA     = 5
MPI         = 8
IsPVM       = (TARGET.JobUniverse == $(PVM))
IsMPI       = (TARGET.JobUniverse == $(MPI))
IsVanilla   = (TARGET.JobUniverse == $(VANILLA))
IsStandard      = (TARGET.JobUniverse == $(STANDARD))

NonCondorLoadAvg  = (LoadAvg - CondorLoadAvg)
BackgroundLoad    = 0.3
HighLoad          = 0.5
StartIdleTime     =  5 * $(MINUTE)
```

```
ContinueIdleTime  =  5 * $(MINUTE)
MaxSuspendTime    = 10 * $(MINUTE)
MaxVacateTime     = 10 * $(MINUTE)

KeyboardBusy           = (KeyboardIdle < $(MINUTE))
ConsoleBusy            = (ConsoleIdle  < $(MINUTE))
CPUIdle                = ($(NonCondorLoadAvg) <= $(BackgroundLoad))
CPUBusy                = ($(NonCondorLoadAvg) >= $(HighLoad))
KeyboardNotBusy        = ($(KeyboardBusy) == False)

BigJob   =(TARGET.ImageSize >= (50 * 1024))
MediumJob=(TARGET.ImageSize    >=    (15    *    1024)&&TARGET.ImageSize
<(50*1024))
SmallJob    = (TARGET.ImageSize <  (15 * 1024))

JustCPU          = ($(CPUBusy) && ($(KeyboardBusy) == False))
MachineBusy      = ($(CPUBusy) || $(KeyboardBusy))

RANK             = 0

TESTINGMODE_WANT_SUSPEND      = False
TESTINGMODE_WANT_VACATE       = False
TESTINGMODE_START             = True
TESTINGMODE_SUSPEND           = True
TESTINGMODE_CONTINUE          = True
TESTINGMODE_PREEMPT           = True
TESTINGMODE_KILL              = False
TESTINGMODE_PERIODIC_CHECKPOINT     =True
TESTINGMODE_PREEMPTION_REQUIREMENTS =True
TESTINGMODE_PREEMPTION_RANK = 0

##########################################################################
##  Part 4:  Settings you should probably leave alone:
##  (unless you know what you're doing)
##########################################################################
##########################################################################

LOG         = $(LOCAL_DIR)/log
SPOOL       = $(LOCAL_DIR)/sPool
EXECUTE     = $(LOCAL_DIR)/execute
BIN         = $(RELEASE_DIR)/bin
LIB         = $(RELEASE_DIR)/lib
INCLUDE     = $(RELEASE_DIR)/include
SBIN        = $(RELEASE_DIR)/sbin
LIBEXEC     = $(RELEASE_DIR)/libexec

HISTORY           = $(SPOOL)/history

COLLECTOR_LOG     = $(LOG)/CollectorLog
KBDD_LOG          = $(LOG)/KbdLog
MASTER_LOG        = $(LOG)/MasterLog
NEGOTIATOR_LOG    = $(LOG)/NegotiatorLog
NEGOTIATOR_MATCH_LOG = $(LOG)/MatchLog
SCHEDD_LOG  = $(LOG)/SchedLog
SHADOW_LOG  = $(LOG)/ShadowLog
STARTD_LOG  = $(LOG)/StartLog
STARTER_LOG = $(LOG)/StarterLog

SHADOW_LOCK = $(LOCK)/ShadowLock

COLLECTOR_HOST  = $(CONDOR_HOST)
```

```
NEGOTIATOR_HOST = $(CONDOR_HOST)

SHUTDOWN_GRACEFUL_TIMEOUT     = 1800

RESERVED_DISK           = 5

RESERVE_AFS_CACHE = False


##  Daemon-specific settings:

##  condor_master

MASTER                  = $(SBIN)/condor_master
STARTD                  = $(SBIN)/condor_startd
SCHEDD                  = $(SBIN)/condor_schedd
KBDD                    = $(SBIN)/condor_kbdd
NEGOTIATOR              = $(SBIN)/condor_negotiator
COLLECTOR               = $(SBIN)/condor_collector
STARTER_LOCAL           = $(SBIN)/condor_starter

MASTER_ADDRESS_FILE = $(LOG)/.master_address

PREEN             = $(SBIN)/condor_preen

PREEN_ARGS              = -m -r

PREEN_INTERVAL                  = 86400

PUBLISH_OBITUARIES              = True

OBITUARY_LOG_LENGTH             = 20

START_MASTER                    = True

START_DAEMONS                   = True

MASTER_UPDATE_INTERVAL          = 300

MASTER_CHECK_NEW_EXEC_INTERVAL      = 300

MASTER_NEW_BINARY_DELAY = 120

SHUTDOWN_FAST_TIMEOUT           = 120

:
MASTER_BACKOFF_FACTOR           = 2.0


MASTER_BACKOFF_CEILING          = 3600

MASTER_RECOVER_FACTOR           = 300

##  condor_startd
```

```
STARTER_LIST      = STARTER, STARTER_PVM, STARTER_STANDARD
STARTER           = $(SBIN)/condor_starter
STARTER_PVM       = $(SBIN)/condor_starter.pvm
STARTER_STANDARD  = $(SBIN)/condor_starter.std
STARTER_LOCAL     = $(SBIN)/condor_starter

STARTD_ADDRESS_FILE     = $(LOG)/.startd_address

POLLING_INTERVAL         = 5

UPDATE_INTERVAL          = 300

MATCH_TIMEOUT            = 300

KILLING_TIMEOUT   = 30

RunBenchmarks : False

STARTD_HAS_BAD_UTMP = True

CONSOLE_DEVICES   = mouse, console

MY_CONSOLE_DEVICES = "$(CONSOLE_DEVICES)"

STARTD_ATTRS = MY_CONSOLE_DEVICES, RunBenchmarks, UPDATE_INTERVAL

##--------------------------------------------------------------------
##  condor_schedd
##--------------------------------------------------------------------

SHADOW_LIST = SHADOW, SHADOW_PVM, SHADOW_STANDARD
SHADOW            = $(SBIN)/condor_shadow
SHADOW_PVM        = $(SBIN)/condor_shadow.pvm
SHADOW_STANDARD   = $(SBIN)/condor_shadow.std

SCHEDD_ADDRESS_FILE     = $(LOG)/.schedd_address

SCHEDD_INTERVAL   = 300

JOB_START_DELAY   = 2

ALIVE_INTERVAL          = 300

MAX_SHADOW_EXCEPTIONS   = 5

SHADOW_SIZE_ESTIMATE    = 1800

SHADOW_RENICE_INCREMENT = 10

NEGOTIATE_ALL_JOBS_IN_CLUSTER = True

QUEUE_SUPER_USERS = root, condor

##--------------------------------------------------------------------
##  condor_starter
##--------------------------------------------------------------------

JOB_RENICE_INCREMENT    = 10

STARTER_LOCAL_LOGGING   = TRUE
```

```
SOFT_UID_DOMAIN   = FALSE

##--------------------------------------------------------------------
##  condor_submit
##--------------------------------------------------------------------

MACHINE = "$(FULL_HOSTNAME)"
SUBMIT_EXPRS = MACHINE

DEFAULT_IO_BUFFER_SIZE = 524288

DEFAULT_IO_BUFFER_BLOCK_SIZE = 32768

##--------------------------------------------------------------------
##  condor_preen
##--------------------------------------------------------------------

PREEN_ADMIN       = $(CONDOR_ADMIN)

VALID_SPOOL_FILES =    job_queue.log,    job_queue.log.tmp,    history,
\Accountant.log,         Accountantnew.log,         \local_univ_execute,
.quillwritepassword

INVALID_LOG_FILES = core

##--------------------------------------------------------------------
##  condor_credd credential managment daemon
##--------------------------------------------------------------------

CREDD                     = $(SBIN)/condor_credd

CREDD_ADDRESS_FILE      = $(LOG)/.credd_address
```

- **Fichero de configuración local**

Una vez que se han modificado los parámetros del archivo de configuración global del Pool, debe procederse a la modificación de los archivos de configuración local del Central Manager situado en */usr/local/condor/etc/maquina1.local*. Del mismo modo que en el fichero de configuración global se han señalizado los cambios realizados, y aquellas variables que deben tenerse en cuenta.

[condor@maquina1 etc]$ *emacs maquina1.local*

Del mismo modo que en el fichero de configuración global se han señalizado los cambios realizados, y aquellas variables que deben tenerse en cuenta.

```
##########################################################################
##
##  condor_config.local.central.manager
##########################################################################

COLLECTOR_NAME          = PROYECTO-UPCT-CONDOR

DAEMON_LIST   = MASTER, COLLECTOR, NEGOTIATOR, STARTD, SCHEDD

COLLECTOR     = $(SBIN)/condor_collector
NEGOTIATOR    = $(SBIN)/condor_negotiator

##--------------------------------------------------------------------
##  condor_collector
##--------------------------------------------------------------------
```

```
CLASSAD_LIFETIME  = 900

MASTER_CHECK_INTERVAL   = 10800

CLIENT_TIMEOUT          = 30

QUERY_TIMEOUT           = 60

##--------------------------------------------------------------------
##  condor_negotiator
##--------------------------------------------------------------------

NEGOTIATOR_INTERVAL     = 300

NEGOTIATOR_TIMEOUT      = 30

PRIORITY_HALFLIFE = 86400

NICE_USER_PRIO_FACTOR = 10000000

MAX_ACCOUNTANT_DATABASE_SIZE = 1000000

REMOTE_PRIO_FACTOR = 10000

NEGOTIATOR_SOCKET_CACHE_SIZE = 16
```

3.3.5 Puesta en marcha de los demonios Condor.

Una vez terminada la instalación, deben reiniciarse las máquinas, empezando por el Central Manager, para que los cambios se hagan efectivos.

Para poner en marcha los demonios de Condor hay que ejecutar en todas las máquinas el demonio **condor_master**, empezando por el Central Manager. Dicho demonio se encuentra en la ruta */usr/local/condor/sbin/condor_master*, aunque gracias a las modificaciones que se han realizado, se puede ejecutar **condor_master** desde cualquier ruta.

[condor@maquina1 sbin]$ *condor_master*

[condor@maquina2 sbin]$ *condor_master*

[condor@maquina3 sbin]$ *condor_master*

[condor@maquina4 sbin]$ *condor_master*

La función básica del condor_master es la de asegurarse de que los demás demonios de Condor están ejecutándose. El master vigila a los demonios, los reinicia si se paran, y periódicamente comprueba si se han instalado nuevos binarios, y reinicia los demonios si dichos binarios les afectan. ara comprobar que los demonios de Condor están funcionando correctamente se ejecuta el comando que se muestra en la siguiente figura:

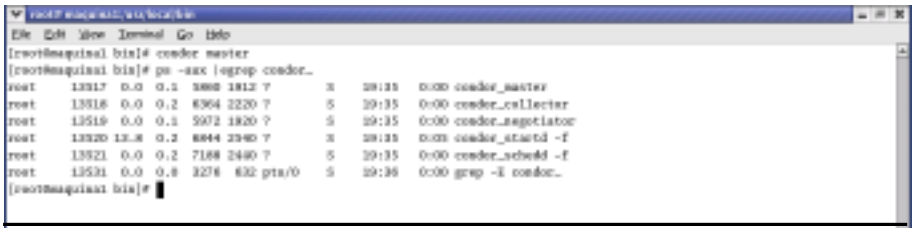


Figura 3. 25 Demonios Condor del Central Manager

En la Figura 3.3.5.1 se observan los demonios del Central Manager. os demonios de las máquinas del Pool de este proyecto son:

Máquina: Función	Demonios
1ª: Central Manager, Submit, Execute	condor_master, condor_collector, condor_negotiator, condor_startd, condor_schedd
2ª: Submit, Execute	condor_master, condor_schedd, condor_startd
3ª: Submit, Execute	condor_master, condor_schedd, condor_startd
4ª: Submit	condor_master, condor_startd

Tabla 3. 6 Demonios de las máquinas que forman parte del Pool

Una vez comprobado que los demonios funcionan correctamente, ya es posible comenzar a ejecutar comandos de Condor, y lanzar tareas.

Capítulo 4
Lanzar tareas: submitting, queueing, executing.

4.1 Fichero de descripción submit

Una tarea es lanzada para su ejecución en Condor usando el comando condor_submit. condor_submit toma como argumento el nombre de un fichero llamado fichero de descripción submit. Ese fichero contiene comandos y palabras clave para controlar el encolamiento de las tareas. En el fichero de descripción submit Condor encuentra todo lo que necesita conocer sobre la tarea. Incluye términos como el nombre del ejecutable, el directorio de trabajo inicial y argumentos de la línea de comandos del programa, todos ellos dentro del fichero de descripción submit. Con toda esta información Condor crea una ClassAd y trabaja para la ejecución de la tarea.

El contenido del fichero de descripción submit puede recortar tiempo para los usuarios de Condor. Es fácil si lo que se desea es lanzar una misma tarea muchas veces: Por ejemplo, si se ejecuta el mismo programa 500 veces con 500 entradas de datos distintas, hay que organizar los ficheros de datos de manera que cada ejecución lea su propio fichero de entrada, y que cada ejecución escriba en su propio fichero de salida. Cada ejecución individual puede tener su propio directorio de trabajo inicial, stdin, stdout, stderr, argumentos de la línea de comandos, y encapsulación. Un programa que directamente abre su propio fichero leerá los nombres de ficheros ya sea desde stdin o desde la línea de comandos. Un programa que abre un fichero estático necesitará usar un subdirectorio separado para la salida de cada ejecución.

4.1.1 ClassAd

Antes de aprender a lanzar una tarea, es importante comprender como Condor asigna recursos. Comprendiendo la única forma de asignar las tareas lanzadas con las máquinas es la clave para obtener la mejor ejecución posible.

Condor simplifica el lanzamiento de la tarea actuando el manejador de ClassAds. Las máquinas indican que atributos poseen, esperando atraer a una tarea como por ejemplo, cantidad de memoria RAM, tipo y velocidad de la CPU, memoria virtual, carga media, etc. Los que lanzan las tareas indican especificaciones sobre lo que quieren que cumpla una máquina para poder ejecutar su tarea.

El ClassAd de la máquina también indica sobre que condiciones desea ejecutar una tarea, y que tipo de tareas prefiere. Estos atributos de las políticas pueden reflejar las preferencias individuales por las cuales los diferentes propietarios han permitido que sus máquinas formen parte del Pool. Debe advertirse que la máquina sólo puede ejecutar tareas mientras que no haya actividad en la máquina. También puede indicarse una preferencia para ejecutar tareas lanzadas por uno mismo o por otra máquina.

Del mismo modo, cuando se lanza una tarea, se especifica una ClassAd con sus requisitos y preferencias. La ClassAd incluye el tipo de máquina que se desea

usar. Por ejemplo, quizás se esté buscando la representación disponible en punto flotante más rápida. Puede desearse mostrar el número de máquinas disponibles basadas en una representación en punto flotante. O quizás puede desearse que la máquina tenga un mínimo de 128 Mbytes de memoria. O puede que sólo se quiera tomar la primera de las máquinas que se posea. Estos atributos y requisitos de la tarea se unen dentro de la ClassAd de una tarea.

Condor juega el papel de manejador, leyendo de manera continua todos los ClassAds de las tareas, uniendo y organizando las tareas con las máquinas. Condor se asegura de que todos los requisitos de las ClassAds se satisfacen.

Para poder mostrar las ClassAds que hay en Condor, se deben emplear los siguientes parámetros:

Comando	Descripción
Condor_status – available	Muestra sólo máquinas que están dispuestas a ejecutar tareas ahora
Condor_status –run	Muestra sólo máquinas que están ejecutando tareas
Condor_status –l	Hacen un listado de las ClassAds de todas las máquinas del Pool.

Tabla 4. 1 Comandos para visualizar los ClassAd

4.1.2 Requirements & Rank

Los comandos de Requirements y rank del fichero de descripción submit son poderosos y flexibles. Ambos necesitan ser especificados como expresiones válidas para las ClassAds de Condor, sin embargo hay valores por defecto que son dados por el programa condor_submit si no se definen en el fichero de descripción submit.

Los operadores de comparación (<, >, <=, >=, y ==) comparan cadenas de caracteres. Los operadores de comparación especiales =?= y =!= hacen comparaciones especiales.

Los atributos de las ClassAds permitidos son aquellos que forman parte de una máquina o de la ClassAd de una tarea. Para poder visualizar todos los atributos ClassAd de todas las máquinas del Pool de Condor, hay que ejecutar condor_status –l. El argumento –l de condor_status indica que hay que mostrar todos los atributos ClassAd de todas las máquinas. Los atributos ClassAd de las tareas que hay en la cola se visualizan mediante condor_q –l. El conjunto de ambas todos los atributos permitidos con los que se cuenta.

Para ayudar a entender cuales de estos atributos son significativos, los descriptores buscan los atributos que son comunes para todos los ClassAd de las máquinas. Hay que recordar que debido a que los ClassAd son flexibles, las máquinas incluirán atributos específicos adicionales para la instalación y las políticas de la máquina. A continuación se muestra una tabla con los ClassAds más significativos:

Atributo ClassAd	Descripción
Activity	Cadena que describe la actividad de trabajo de Condor en la máquina. Puede tener uno de los siguientes valores:Idle, Busy, Suspended, Vacating, Killing, Benchmarking.
Arch	Cadena que indica la arquitectura de la máquina. En el caso de este proyecto se empleó la cadena “INTEL” que describe a un Intel x86.
Console Idle	Número de segundos transcurridos desde que se detectó actividad de la consola (ya sea ratón o teclado).

FileSystemDomain	Nombre de dominio configurado por el administrador de Condor que describe a un clúster de máquinas en el que todas las máquinas acceden de la misma manera uniforme, ya sea vía NFS o AFS. Es útil para las tareas Vanilla con acceso a ficheros remotos.
Machina	Una cadena que define el hostname de la máquina.
OpSys	Cadena que describe el sistema operativo de la máquina. En este proyecto se han empleado “LINUX” para Redhat y “WINNT51” para Windows XP.
Requirements	Un booleano, que cuando se evalúa a través del contexto ClassAd o de una tarea ClassAd, debe ser evaluado a TRUE antes de que Condor pueda usar la máquina.
JobId	El identificador de la tarea, que se puede observar por condor_q en la máquina donde se lanzó la tarea.
Cmd	La ruta y el nombre del fichero donde se debe ejecutar la tarea.
ImageSize	Tamaño de la imagen de la tarea en Kbytes de memoria.
JobUniverse	Entero que indica el universo donde ejecutar la tarea.
Owner	Cadena que describe el usuario que lanzó la tarea.
TransferIn	Si es True, entonces la entrada de la tarea se transfiere desde la máquina que lanzó la tarea a la máquina remota. El nombre del fichero que se transfiere se da en el atributo In.
TransferOut	Si es true, entonces la salida de la tarea se transfiere desde la máquina remota de regreso a la máquina que lanzó la tarea. El nombre del fichero a transmitir se da en el atributo Out.

Tabla 4. 2 Descripción de los atributos más significativos de los ClassAd

Expresiones Rank

Cuando se consideran las relaciones entre una tarea y una máquina, la expresión Rank se emplea para elegir una de las máquinas entre todas las existentes que satisfaga las necesidades de la tarea, y estén disponibles para el usuario, teniendo en cuenta las prioridades y el rango de la tarea. Las expresiones Rank, simples o complejas, definen un valor numérico que expresa preferencias.

La expresión Rank de la tarea se evalúa a uno de estos tres valores: UNDEFINED, ERROR, o un valor en punto flotante. Si es un valor en punto flotante, la mejor elección será aquella con el mayor valor positivo. Si no se da ninguna expresión Rank, entonces Condor sustituye el valor por defecto por 0.0. Si adquiere el valor UNDEFINED o ERROR, también se usa el valor 0.0. Por lo tanto, a la máquina se la sigue considerando como válida para la asignación de la tarea, pero no hay ningún valor rank que considerar por encima de otro. Un ejemplo de este tipo de expresión sería:

```
Rank=HAS_MATLAB
```

En este caso Condor sólo buscará aquellas máquinas que posean Matlab.

4.1.3 Lanzar tareas empleando o no un Sistema de ficheros Distribuido

Caso RedHat: Uso de un sistema de ficheros distribuido.

Si las tareas de los universos Vanilla, Java, Paralelo (o MPI) se lanzan sin usar un mecanismo de transmisión de ficheros, Condor debe usar un sistema de ficheros

distribuidos para acceder a los ficheros de entrada y salida. En este caso la máquina debe de ser capaz de acceder a los ficheros de información desde cualquier máquina en la que pueda ejecutarse de manera potencial.

Condor permite a los usuarios asegurarse de que sus tareas tienen acceso al sistema de ficheros, usando los atributos ClassAds FileSystemDomain y UidDomain. Este atributo especifica que máquinas poseen acceso al mismo sistema de ficheros. Todas las máquinas que poseen los mismos directorios compartidos se consideran pertenecientes al mismo sistema de ficheros. De manera similar, todas las máquinas que poseen la misma información de usuario son considerados parte del mismo dominio UID. Por ello si un Pool debe tener acceso a un sistema de ficheros distribuido, el administrador del Pool deben de configurar de manera correcta Condor de forma que todas las máquinas que comparten los mismos ficheros posean el mismo FileSystemDomain en el archivo de configuración. De la misma manera todas las máquinas que posean información común de usuario debe ser configurado con el mismo UidDomain del archivo de configuración.

Cuando una máquina depende de un sistema de ficheros distribuido, Condor emplea la expresión Requirements para asegurarse de que la máquina se ejecuta en el correcto UidDomain y el correcto FileSystemdomain. En este caso la expresión Requirements por defecto especifica que la tarea debe ejecutarse en una máquina con el mismo UidDomain y FileSystemDomain de la máquina donde se lanzó la tarea. Este valor por defecto es casi siempre el correcto. Sin embargo en un Pool con múltiples UidDomain y FileSystemDomain, el usuario necesita especificar una expresión Requirements para poder ejecutar las tareas en las máquinas adecuadas.

Caso Windows: Lanzar tareas sin un sistema de archivos distribuido. Empleo del mecanismo de transferencia de ficheros.

Este mecanismo lo emplea el usuario cuando éste lanza una tarea. Condor transferirá todos los ficheros necesarios para la ejecución de una tarea desde la máquina donde la tarea fue lanzada, a un directorio de trabajo temporal en la máquina donde se ejecuta la tarea. Condor ejecuta la tarea y transfiere la salida de regreso a la máquina que lanzó la tarea. Hay diferentes valores por defecto del sistema de transferencia de ficheros dependiendo del Universo y de la plataforma:

- Para las tareas del Universo Standard la existencia de un sistema de ficheros distribuido no es relevante. El acceso a los ficheros (tanto de entrada como de salida) se realiza a través del mecanismo de llamada a sistema remoto (RPC).
- Para las tareas del Universo Vanilla, java, MPI, y Universo paralelo, el acceso a los ficheros, a través de un sistema de archivos distribuido se presume por defecto en las máquinas UNIX. Si no hay ningún sistema de ficheros, el mecanismo de transferencia de ficheros debe ser especificado de manera específica. Cuando se lanza una tarea desde una máquina Windows, Condor asume lo contrario: no hay acceso a un sistema de ficheros distribuido y activa el mecanismo de transferencia por defecto.
- Para el Universo Grid las tareas son ejecutadas en máquinas remotas, por lo que nunca hay un sistema de ficheros compartido entre las máquinas.
- Para el Universo PVM los ficheros que se transfieren no son otros que el ejecutable del **master** y los ficheros dados como comandos de entrada, salida y error no se soportan.
- Para el Universo Scheluder Condor se usa en la máquina desde la que se lanza la tarea.

Para activar el mecanismo de transferencia de ficheros, dos comandos deben de situarse en el fichero de descripción submit:

- **Should_transfer_files**: especifica cuando Condor debe transferir ficheros de entrada desde la máquina que lanzó la tarea, a la máquina remota que la ejecuta. También especifica cuando los ficheros de salida son enviados de regreso a la máquina que lanzó la tarea. El comando puede tener tres valores posibles:
  - YES: Condor siempre transfiere los ficheros de entrada y salida.
  - IF_NEEDED: Condor transfiere ficheros si a la tarea se le asigna una máquina con un FileSystemDomain distinto a la máquina que lanzó la tarea.
  - NO: Deshabilita el sistema de transferencia de ficheros.
- **When_to_transfer_output**: este comando indica cuando deben de mandarse de regreso los ficheros de salida después de que la tarea sea ejecutada en la máquina remota. Sus posibles valores son:
  - ON_EXIT: Condor envía los ficheros de salida a la máquina que lanzó la tarea, cuando la tarea acaba por sí misma.
  - ON_EXIT_OR_EVICT: Condor siempre realizará la transferencia, si la tarea se completa por sí misma, si la máquina recibe una tarea con mayor prioridad, abandona la máquina, o es eliminada. Cuando las tareas se completan, la salida se manda de regreso a la máquina que lanzó la tarea. En los otros casos, los ficheros son transmitidos de regreso a la vez que abandonan la máquina. Estos ficheros son enviados por el directorio dado por la variable SPOOL del fichero de configuración.
- **Transfer_input_files**: si el mecanismo de transmisión de ficheros está activado, Condor enviará los siguientes ficheros, antes de que la tarea sea ejecutada en la máquina remota: el ejecutable, la entrada definida por el comando input, y los ficheros jar en el caso de que se emplee el Universo Java. Si la tarea requiere otro tipo de entrada Condor deberá emplear este comando. Estos ficheros son empleados en el mismo directorio de trabajo temporal, que el ejecutable de la tarea. Para restringir los ficheros que son enviados de regreso, habrá que especificar una lista exacta con el comando **transfer_output_files**.

4.1.4 Variables de entorno

El entorno en el que se ejecuta una tarea normalmente contiene información que es potencialmente útil para la tarea. Condor permite a un usuario referenciar y establecer las variables de entorno para una tarea o un cluster de tareas. A través de un fichero de descripción submit, el usuario debe definir las variables de entrono para el entorno de la tarea usando el comando **enviroment**.

El entorno de la máquina que lanzó la tarea se puede copiar dentro de la ClassAd de la tarea.

4.2 Tipos de tareas

4.2.1 Tareas Dagman

Un grafo acíclico directo (DAG) se puede usar para representar un conjunto de cálculos, donde la entrada, la salida, o la ejecución de una o más tareas es dependiente de una u otra tarea. Las tareas son nodos (vértices) en el grafo, y los arcos identifican las dependencias. Condor encuentra máquinas para la ejecución de los programas, pero no establece la planificación de dichos programas basados en dependencias. DAGMAN es un controlador para la ejecución de programas. DAGMAN lanza las tareas en Condor en el orden representado por el grafo, y procesa el resultado. Un fichero de entrada DAG describe el grafo, y además los ficheros de descripción submit son usados por DAGMAN cuando lanzan tareas para ser ejecutadas por Condor.

DAGMAN es asimismo ejecutado como una tarea del Universo Scheluder a través de Condor. Del mismo modo que DAGMAN lanza tareas, controla los ficheros log para hacer cumplir el orden requerido por el DAG. DAGMAN también es responsable de catalogar, recuperar e informar al conjunto de tareas lanzadas en Condor. Un nodo en un DAG debe abarcar más de un programa lanzado para su ejecución sobre Condor. Una limitación que posee es que todas las tareas del mismo clúster deben usar el mismo fichero log.

Una tarea DAGMAN organiza y lanza las tareas a través de los nodos a Condor, las cuales son definidas para tener éxito o fallar dependiendo de sus valores de retorno. Dicho éxito o fallo se propaga de manera bien definida a nivel del nodo a través de DAG. El fallo de una sola tarea a través de un clúster con múltiples tareas causan que el conjunto entero de las tareas fallen. Todas las tareas del clúster son eliminadas de manera inmediata. Cualquier nodo del DAG está definido para tener éxito o fallar, basado en los valores devueltos por el script PRE, las tareas del clúster, y/o los script POST.

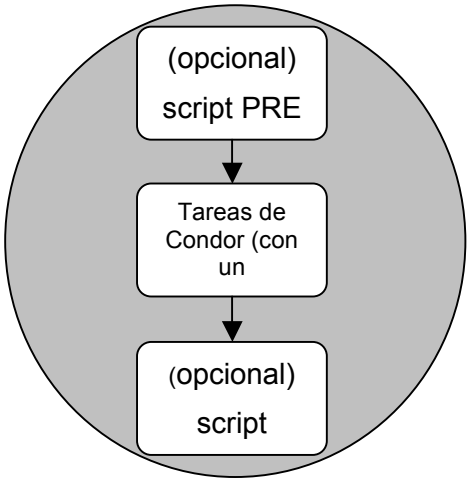


Figura 4. 1 Nodo de una tarea DAGMAN

4.2.2 Tareas Matlab

Matlab es al mismo tiempo un entorno y un lenguaje de programación. Uno de sus puntos fuertes es el hecho de que el lenguaje de Matlab permite construir nuestras propias herramientas reusables. Podemos fácilmente crear nuestras propias funciones y programas especiales (conocidos como archivos .m) en código Matlab. Los podemos

agrupar en Toolbox: colección especializada de archivos .m para trabajar en clases particulares de problemas.

La manera más fácil de visualizar Matlab es pensar en él como en una calculadora totalmente equipada, aunque en realidad, ofrece muchas más características y es mucho más versátil que cualquier calculadora. Matlab es una herramienta para hacer cálculos matemáticos. Es una plataforma de desarrollo de aplicaciones, donde conjuntos de herramientas inteligentes para resolución de problemas en áreas de aplicación específica, a menudo llamadas toolboxes, se pueden desarrollar con una facilidad relativa.

4.2.3 Tareas en C y Fortran

- **Tareas en C**

C es un lenguaje de programación relativamente minimalista. Uno de los objetivos de diseño de este lenguaje fue que sólo fueran necesarias unas pocas instrucciones en lenguaje máquina para traducir cada elemento del lenguaje, sin que hiciera falta un soporte intenso en tiempo de ejecución. Es muy posible escribir C a bajo nivel de abstracción; de hecho, C se usó como intermediario entre diferentes lenguajes. En parte a causa de ser de relativamente bajo nivel y de tener un modesto conjunto de características, se pueden desarrollar compiladores de C fácilmente. En consecuencia, el lenguaje C está disponible en un amplio abanico de plataformas (seguramente más que cualquier otro lenguaje). Además, a pesar de su naturaleza de bajo nivel, el lenguaje se desarrolló para incentivar la programación independiente de la máquina. Un programa escrito cumpliendo los estándares e intentando que sea portable puede compilarse en muchos computadores.

C se desarrolló originalmente (conjuntamente con el sistema operativo Unix, con el que ha estado asociado mucho tiempo) por programadores para programadores. Sin embargo, ha alcanzado una popularidad enorme, y se ha usado en contextos muy alejados de la programación de sistemas, para la que se diseñó originalmente.

- **Tareas Fortran**

El lenguaje fue diseñado tomando en cuenta que los programas serían escritos en tarjetas perforadas de 80 columnas. Así por ejemplo, las líneas debían ser numeradas y la única alteración posible en el orden de ejecución era producida con la instrucción goto. Estas características han evolucionado de versión en versión. Las versiones actuales contienen subprogramas, recursión y una variada gama de estructuras de control.

Lo que fue la primera tentativa de proyección de un lenguaje de programación de alto nivel, tiene una sintaxis considerada arcaica por muchos programadores que aprenden lenguajes más modernos. Es difícil escribir un bucle "for", y errores en la escritura de sólo un carácter pueden llevar a errores durante el tiempo de ejecución en vez de errores de compilación, en el caso de que no se usen las construcciones más frecuentes. Algunas de las versiones anteriores no poseían facilidades que son consideradas como útiles en las máquinas modernas, como la colocación dinámica de memoria. Se debe tener en cuenta que la sintaxis de Fortran fue afinada para el uso en trabajos numéricos y científicos y que muchas de sus deficiencias han sido abordadas en revisiones más recientes del lenguaje. Por ejemplo, Fortran 95 posee comandos mucho más breves para efectuar operaciones matemáticas con matrices y dispone de tipos. Esto no sólo mejora mucho la lectura del programa sino que además aporta información útil al compilador. Por estas razones Fortran no es muy usado fuera de los campos de la informática y el análisis numérico, pero permanece como el lenguaje a escoger para desempeñar tareas de computación numérica de alto rendimiento.

4.3 Proceso de lanzamiento de tareas.

4.3.1 Tareas Dagman

4.3.1.1 Fichero de entrada que describe el DAG

Describe siete tipos de ítems:

1. Una lista de nodos en el DAG que causa el lanzamiento de una o más tareas de Condor. Cada entrada sirve para nombrar un nodo y se refiere a un fichero de descripción submit.
2. Una lista de nodos en el DAG que causa el envío de datos de tareas en cola. Cada entrada se emplea para nombrar un nodo y especificar el fichero de descripción submit Stork.
3. Cualquier proceso que necesita situarse en un nodo de condor o una tarea Stork.
4. Una descripción de las dependencias del DAG.
5. El número de veces que se reintenta la ejecución de un nodo, si un nodo del DAG falla.
6. Cualquier definición de macro asociada a un nodo.
7. Un valor del nodo de salida que causa que todo el DAG falle.

Pueden incluirse comentarios en el fichero de entrada del DAG, poniendo de antemano el carácter #.

Comandos del fichero de entrada:

JOB

El primer conjunto de líneas en el fichero de entrada DAG describe a cada una de las tareas que aparecen en el DAG. Cada nodo que va a ser manejado por Condor se describe como una sola línea que comienza con la palabra clave JOB. La sintaxis empleada para cada entrada JOB es:

JOB nombre_de_la_tarea Nombre_del_fichero_de_descripcion_submit[done]

Una entrada JOB describe el nombre de la tarea del fichero de descripción submit. El nombre de la tarea sólo identifica nodos a través de los ficheros de entrada de DAGMAN y los mensajes de salida.

El parámetro DONE identifica una tarea que ha sido completada. Es útil en situaciones donde los usuarios desean verificar resultados, sin la necesidad de ejecutar todas las tareas del grafo. La macro DONE también se emplea cuando ocurre un error que hace que el grafo finalice sin completarse. DAGMAN genera un DAG Rescue, un fichero de entrada DAG que puede usarse para reiniciar y completar un DAG sin reejecutar los nodos completos, sólo la parte que falló.

DATA

Identifica una tarea que debe ser manejada por el servidor de datos Stork. La sintaxis empleada para cada entada DATA es:

DATA JobName SubmitDescriptionFileName [DONE]

Una entrada DATA traza el mapa del nombre de una tarea con un fichero de descripción submit Store. En el resto de los aspectos el comando DATA es idéntico al del comando JOB.

SCRIPT

Identifica los procesamientos realizados antes de que una tarea del DAG se lance a Condor o a Stork para su ejecución, o después de que una tarea del DAG complete su ejecución. Los procesos que se realicen antes de que se ejecute una tarea se llama script PRE. Los procesos que se producen después de que las tareas completen su ejecución se llaman script POST.

SCRIPT PRE JobName ExecutableName[arguments]

SCRIPT POST JobName ExecutableName[arguments]

Las palabras PRE o POST especifican el cronometraje de cuando debe ejecutarse el script. El parámetro JobName especifica el nodo en el que se ejecuta el script. El ExecutableName especifica el script que va a ser ejecutado.

PARENT..CHILD

Especifican las dependencias del DAG. Los nodos son padres o hijos en el DAG. Un nodo padre debe completarse de manera satisfactoria, antes de que uno de sus hijos comience. Un nodo hijo sólo empezará una vez que todos sus padres hayan finalizado. La sintaxis es:

PARENT ParentJobName…CHILD ChildJobName…

RETRY

Es una manera de reintentar los nodos fallidos

RETRY JobName NumberOfRetries[UNLESS_EXIT<value>]

Donde JobName identifica el nodo y NumberOfRetries es un entero, que indica el número de reintentos del nodo después del fallo

VARS

Se usa para definir macros. Su sintaxis es la que sigue:

VARS JobName macroname=”string”[macroname=”string”…]

Esta macro puede ser usada en un fichero de descripción submit

ABORT-DAG-ON

Elimina el DAG por completo si un nodo dado, devuelve un valor específico. Su sintaxis es:

ABORT-DAG-ON JobName AbortExitValue [return DAGReturnValue]

Si el nodo especificado por el parámetro JobName retorna el valor específico AbortExitValue, el DAG aborta su ejecución. Un DAG que se aborta, a diferencia de un nodo fallido, hace que todos los nodos del DAG paren su ejecución inmediatamente. Esto incluye eliminar las tareas de los nodos que están ejecutándose. Un nodo fallido permite que DAGMAN continúe ejecutándose, aunque no podrán realizarse más procesos debido a las dependencias. Este comando también elimina las dependencias entre los nodos. El DAG para, aunque el nodo posea reintentos.

4.3.1.2 Fichero de descripción Submit

Cada nodo del DAG usa un único fichero de descripción submit. Una limitación clave, es que cada fichero de descripción submit de Condor debe lanzar tareas descritas por un único número de clúster.

Todas las tareas de todos los nodos generan ficheros de descripción log. Sin embargo el rendimiento puede verse afectado, por lo que es mejor reducir el número de ficheros log.

4.3.1.3 Tarea Propuesta

El fichero de descripción submit de la tarea propuesta es el siguiente:

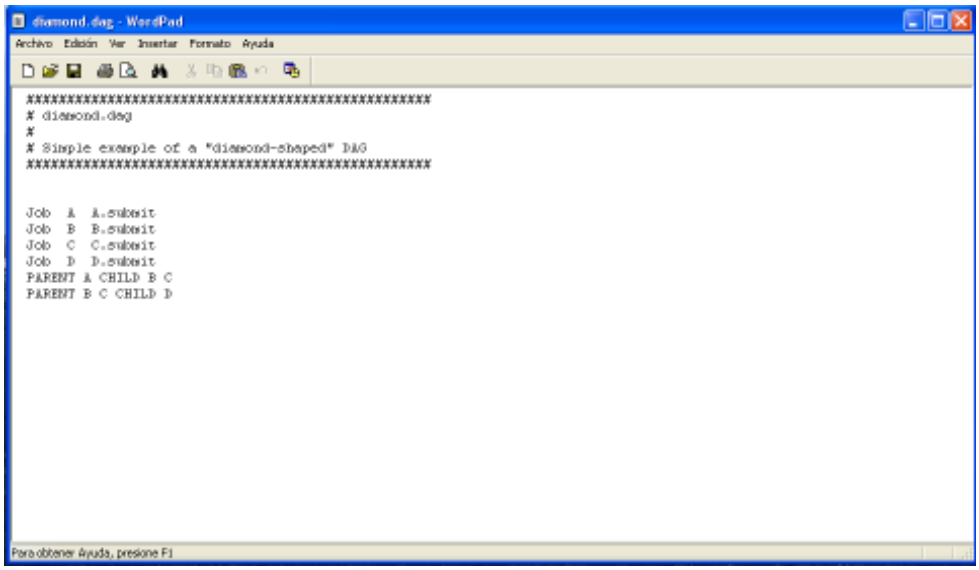
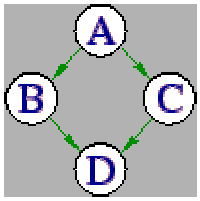


Figura 4. 2 Fichero de descripción submit de Dagman

Dicho fichero indica la siguiente relación:



Este DAG posee cuatro tareas: A. B, C, D. Las tareas poseen las siguientes dependencias: A es el padre, B y C son hijos de A, y D es hijo de B y C.

Los ficheros de descripción submit de las tareas independientes son los siguientes:

TAREA A

```
####################################################
# Fichero submit para la tarea A
####################################################

Universe   = vanilla
Executable = random.condor
output     = A.out
log        = diamond.log
Queue
```

En esta tarea, el ejecutable random.condor se refiere a un programa en C que genera un número aleatorio divisible por 2, y escribe el resultado en el archivo A.out,

descrito en la entrada output. Este programa se ejecuta en el universo Vanilla. Las características de la ejecución se muestran en el archivo log diamond.log, y la tarea se pone en cola gracias al comando Queue.

TAREA B

```
##################################################
# Fichero submit para la tarea B
#
# half.condor lee un número desde STDIN,
# lo divide por 2, y muestra el resultado en STDOUT.
##################################################

Universe   = vanilla
Executable = half.condor
input      = A.out
output     = B.out
log        = diamond.log
Queue
```

En esta tarea el ejecutable half.condor, toma una entrada desde el archivo A.out, cuyo contenido es un entero, divide dicho entero por 2, y escribe el resultado en el fichero B.out. Al igual que la tarea anterior, sus características se muestran en el archivo diamond.log, y la tarea se pone en cola con el comando Queue.

TAREA C

```
##################################################
# Fichero submit para la tarea C
##################################################

Universe   = vanilla
Executable = half.condor
input      = A.out
output     = C.out
log        = diamond.log
Queue
```

Al igual que la tarea B, toma el entero procedente del archivo A.out, lo divide entre 2 mediante el programa en C half.condor, y escribe el resultado en el archivo C.out, por lo que posee la misma salida que B.out. También muestra el proceso de ejecución en el archivo diamond.log y pone la tarea en cola con el comando Queue.

TAREA D

```
##################################################
# Submit file for Job D
##################################################

Universe   =vanilla
Executable =sum.condor
arguments  = B.out C.out
output     = D.out
error      = D.err
log        = diamond.log
Queue
```

La tarea D ejecuta un programa en C denominado sum.c que toma los archivos procedentes de B.out y C.out, los suma y escribe el resultado en la salida D.out, que es el mismo número que el que se creó en el archivo A.out. Si se produce algún error, éste se mostrará en el archivo D.err. Para finalizar, las características de la ejecución se muestran también en el archivo diamond.log y la tarea se pone en cola mediante el comando Queue.

4.3.1.4 Lanzamiento de la tarea

Lanzamiento de la tarea en Windows

Para lanzar la tarea el primer paso es la compilación de los archivos. En Windows no existe un ejecutable condor_compile como en el caso de Redhat. Para realizar la compilación de los archivos en C se ejecutará lo siguiente:

gcc –o random.condor random.c

gcc –o half.condor half.c

gcc –o sum.condor sum.c

Esto compilará los archivos y generará los ejecutables random.condor, half.condor y sum.condor necesarios para poder realizar la tarea descrita. Para poder realizar la tarea en Condor hay que ejecutar:

condor_submit_dag diamond.dag

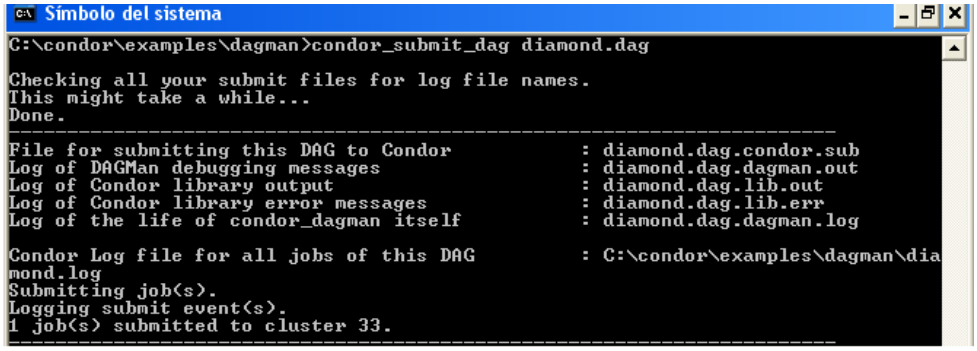


Figura 4. 3 Lanzamiento de la tarea Dagman

Lanzamiento de las tarea en Redhat

La principal diferencia entre el lanzamiento de la tarea en un entorno u otro está en la compilación de los archivos. En Redhat se posee el binario condor_compile que enlaza las librerías de Condor con las tareas a lanzar.

El archivo Makefile para la compilación de las tareas del grafo sería el siguiente:

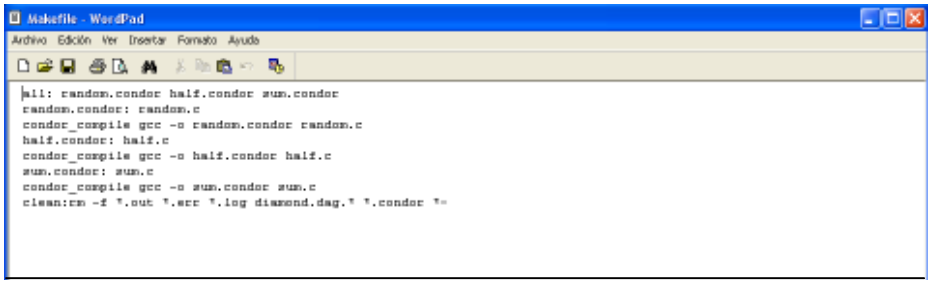


Figura 4. 4 Archivo Makefile de la tarea Dagman

Por lo que para lanzar la tarea se deberá ejecutar lo siguiente:

[condor@maquina2 dagman]$ make

[condor@maquina2 dagman]$ condor_submit_dag diamond.dag

El resultado de la ejecución de estos comandos es el mismo que pudo observarse en la Figura 4.3. Los archivos que se crean se muestran en la figura siguiente:

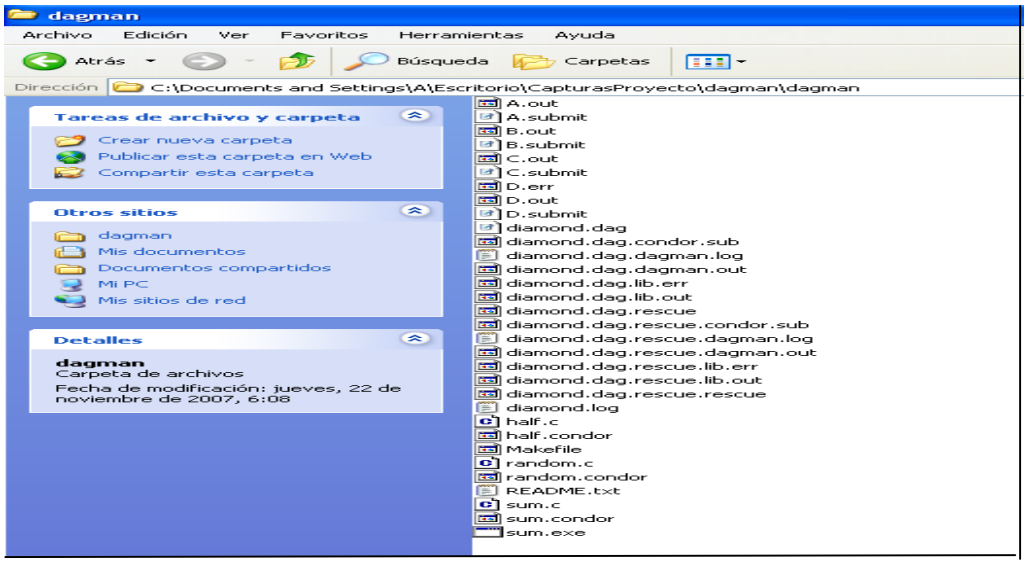


Figura 4. 5 Ficheros creados tras la compilación y ejecución de los archivos Dagman

4.3.2 Tareas Matlab

Antes de comenzar con la explicación sobre el lanzamiento y ejecución de las tareas, es necesario indicar que durante la realización de este proyecto no pudo demostrarse el funcionamiento de este tipo de tareas en Condor.

Tras mucho investigar en libros y foros de Internet, se llegó a la conclusión de que el problema era dependiente de las propias máquinas. Aún así a continuación se explica como se realizaría la creación y lanzamiento de una tarea en Matlab.

4.3.2.1 Tarea propuesta

El primer paso es la creación de un fichero de descripción de la tarea (fichero .m), que contenga la propia tarea a ejecutar. En el caso ejemplo dicha tarea consiste en el cálculo del factorial del número 13, llamándose el archivo *fact.m* que es el que se describe a continuación:

```
factorial(13)
quit
```

El siguiente paso es la creación del archivo ejecutable:

Para el caso de RedHat (matlab.sh):

```
#!/bin/bash
/usr/local/bin/matlab -nojvm -nodisplay -nosplash -r fact.m
```

Para el caso de Windows (matlab.bat):

```
set path=%path%;c:\Archivos de programa\MATLAB\R2007a\bin\win32
MATLAB.exe -nojvm -nodisplay -nosplash -r fact.m
```

Para el caso de Redhat, la ejecución de la tarea se produce de manera directa, pero en el caso de Windows (archivo matlab.bat) es necesario especificar el path de Matlab.

Una vez hecho esto se procede a la creación del fichero de descripción submit llamado *matlab.sub*, que será el que propiamente se lance en Condor. Uno de los puntos calves de estos ficheros es que el único universo posible en el que la tarea podrá lanzarse es el universo Vanilla, ya que Matlab sólo admite este universo para su ejecución:

Para el caso de RedHat:

```
Universe= Vanilla
Requierements= HAS_MATLAB =?= TRUE && OpSys == "LINUX" && Arch ==
"x86"
Enviroment= HOME=/home/condor
Executable=matlab.sh
Arguments=fact
Should_transfer_files=YES
When_to_transfer_output=ON_EXIT
Input=fact.m
Output=Matlab.out
Log=Matlab.log
Error=Matlab.err
Queue
```

Para el caso de Windows:

```
Requierements= HAS_MATLAB =?= TRUE && OpSys == "WINNT51" && Arch ==
"x86"
Enviroment= c:\condor\bin\examples
Executable=matlab.bat
Arguments=fact
Should_transfer_files=YES
When_to_transfer_output=ON_EXIT
Input=fact.m
Output=Matlab.out
Log=Matlab.log
Error=Matlab.err
Queue
```

En ambos archivos se toma como argumento de entrada *Input* el fichero *fact.m*, escribiendo el resultado de la salida *Output* en el fichero *Matlab.out*, y generando los archivos de salida *Matlab.log*, que posee un resumen de los sucesos de la ejecución, y *Matlab.err* que mostrará los fallos en el caso de que hubieran.

Para poder aceptar los ficheros de entrada, es necesario habilitar *Should_transfer_files*, y para poder obtener la salida desde Matlab, debe habilitarse *When_to_transfer_output*, que en este caso al establecerse como *ON_EXIT*, sólo mostrará el resultado tras la finalización completa de la tarea. En *arguments* se encuentra el archivo compilado de fact.m.

La ejecución de tareas Matlab en RedHat y Windows se diferencian en que en este último el ejecutable *Executable* es *matlab.bat,* y para Redhat es *matlab.sh*. Además en la especificación de las características, *Requierements*, que se desean que las máquinas de ejecución posean, en el caso de Redhat hay que poner como sistema operativo *OpSys LINUX*, y en el caso de Windows hay que poner *WINNT51*.

4.3.2.2 Lanzamiento de la tarea

Para lanzar la tarea, los pasos son los mismos en Linux y en Windows: El primer paso es la compilación de la misma que se realiza mediante el siguiente comando:

```
[condor@maquina2 examples]$ mcc -mv fact.m
```

Para lanzar la tarea habrá que ejecutar lo siguiente:

```
[condor@maquina2 examples]$ condor_submit matlab.sub
```

Una vez hecho esto Condor se encargará de gestionar la ejecución de la tarea, y se crearán los archivos explicados anteriormente

4.3.3 Tareas C y Fortran

Las tareas C y Fortran, son las más sencillas de implementar. Sólo se diferencian en el proceso de lanzamiento de la tarea, en el comando que se usa para compilar. Por ello se va a describir el lanzamiento de una tarea en C, indicándose en que puntos de diferencia de Fortran, y en que puntos se diferencian los sistemas operativos.

4.3.3.1 Tarea Propuesta

El primer paso para el lanzamiento de una tarea en C o en Fortran es la creación de la propia tarea. En el caso de ejemplo de C, la tarea loop.c consiste en un bucle que imprime cinco archivos (lanza 5 tareas), que imprimen números del 1 al 200 en dos de los archivos, de uno a 300 en otros dos y de 1 a 500 en el último.

El archivo de descripción submit loop.submit de la tarea es el siguiente:

```
universe=vanilla
requeriments=(Arch=="Intel")&&(OpSys=="WINNT51")
environment=path=c:\Windows\system32
executable  = c:\condor\examples\loop.exe
output            = loop.$(Process).out
error       = loop.$(Process).err
log         = loop.log
arguments   = 200
queue 2
arguments   = 300
queue 2
arguments   = 500
output            = loop.last.out
error       = loop.last.err
queue
```

Para poder lanzar las tareas en Linux habría que cambiar:

```
requeriments=(Arch=="Intel")&&(OpSys=="LINUX")
environment=/home/condor
executable=loop.remote
```

Los parámetros que se muestran son los mismos que se explicaron en las tareas de matlab. Queue 2 hace que 2 tareas del mismo ejecutable se pongan en cola.

4.3.3.2 Lanzamiento de la tarea

Para lanzar la tarea, los pasos son los mismos en Linux y en Windows salvo a la hora de compilar:

Para el caso de Linux:

[condor@maquina2 examples]$ `condor_compile gcc -o loop.remote loop.c`

Si la tarea fuera de Fortran la única diferencia sería a la hora de compilar:

[condor@maquina2 examples]$ `condor_compile g77 -o tarea.remote tarea.f`

Para el caso de Windows:

C:\condor\examples> gcc –o loop.exe loop.c

Para lanzar la tarea habrá que ejecutar lo siguiente tanto en Linux como en Windows:

[condor@maquina2 examples]$ `condor_submit loop.sub`

Una vez hecho esto Condor se encargará de gestionar la ejecución de la tarea, y se crearán los archivos de salida correspondientes.

4.4 Proceso de ejecución de tareas Dagman.

Para poder visualizar la tarea que se lanzó mediante línea de comandos los comandos más útiles son condor_status que nos mostrará el estado de las máquinas que forman parte del Pool, y condor_q que nos mostrará el proceso de ejecución de las mismas:

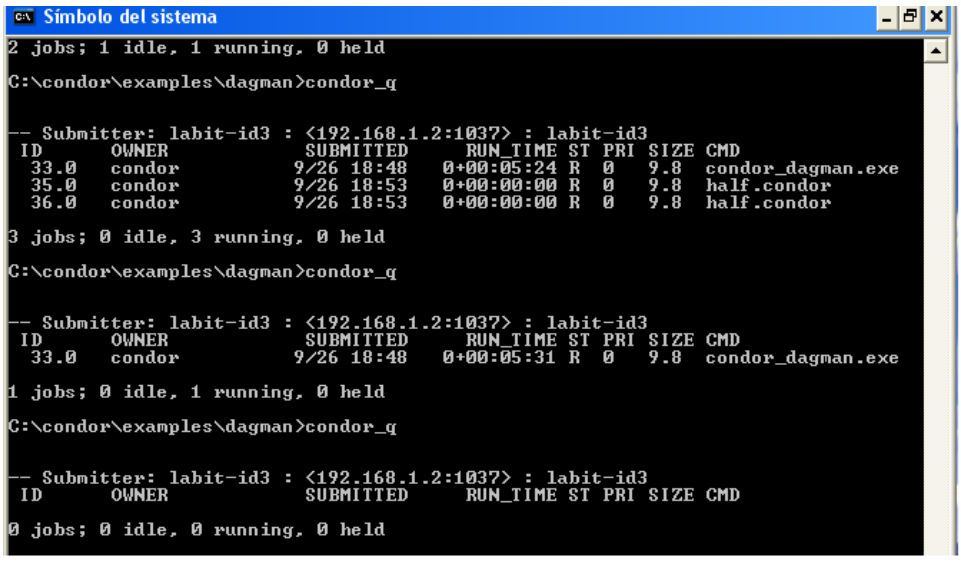


Figura 4. 6 Visualización del estado de las tareas mediante condor_q

Otra manera de poder ver el progreso de las tareas es mediante el archivo diamond.dag.dagman.out, que muestra paso por paso el progreso de una tarea, indicando, si ocurre, cuando se produce un fallo. Un ejemplo de parte de este archivo es el siguiente:

```
9/26 18:48:26 ******************************************************
9/26 18:48:26 ** condor_scheduniv_exec.33.0 (CONDOR_DAGMAN) STARTING
UP
9/26 18:48:26 ** c:\condor\bin\condor_dagman.exe
9/26 18:48:26 ** $CondorVersion: 6.8.4 Feb  1 2007 $
9/26 18:48:26 ** $CondorPlatform: INTEL-WINNT50 $
9/26 18:48:26 ** PID = 1644
9/26 18:48:26 ** Log last touched time unavailable (No such file or
directory)
9/26 18:48:26 ******************************************************
9/26 18:48:26 Using config source: C:\condor\condor_config
9/26 18:48:26 Using local config sources:
9/26 18:48:26    C:\condor/condor_config.local
9/26 18:48:26 DaemonCore: Command Socket at <192.168.1.2:1248>
9/26 18:48:26 DAGMAN_SUBMIT_DELAY setting: 0
9/26 18:48:26 DAGMAN_MAX_SUBMIT_ATTEMPTS setting: 6
9/26 18:48:26 DAGMAN_STARTUP_CYCLE_DETECT setting: 0
9/26 18:48:26 DAGMAN_MAX_SUBMITS_PER_INTERVAL setting: 5
9/26 18:48:26 allow_events (DAGMAN_IGNORE_DUPLICATE_JOB_EXECUTION,
DAGMAN_ALLOW_EVENTS) setting: 114
9/26 18:48:26 DAGMAN_RETRY_SUBMIT_FIRST setting: 1
9/26 18:48:26 DAGMAN_RETRY_NODE_FIRST setting: 0
9/26 18:48:26 DAGMAN_MAX_JOBS_IDLE setting: 0
9/26 18:48:26 DAGMAN_MAX_JOBS_SUBMITTED setting: 0
9/26 18:48:26 DAGMAN_MUNGE_NODE_NAMES setting: 1
9/26 18:48:26 DAGMAN_DELETE_OLD_LOGS setting: 1
9/26 18:48:26 DAGMAN_PROHIBIT_MULTI_JOBS setting: 0
9/26 18:48:26 DAGMAN_ABORT_DUPLICATES setting: 0
9/26 18:48:26 argv[0] == "condor_scheduniv_exec.33.0"
9/26 18:48:26 argv[1] == "-Debug"
9/26 18:48:26 argv[2] == "3"
9/26 18:48:26 argv[3] == "-Lockfile"
9/26 18:48:26 argv[4] == "diamond.dag.lock"
9/26 18:48:26 argv[5] == "-Condorlog"
9/26 18:48:26 argv[6] == "C:\condor\examples\dagman\diamond.log"
9/26 18:48:26 argv[7] == "-Dag"
9/26 18:48:26 argv[8] == "diamond.dag"
9/26 18:48:26 argv[9] == "-Rescue"
9/26 18:48:26 argv[10] == "diamond.dag.rescue"
9/26 18:48:26 DAG Lockfile will be written to diamond.dag.lock
9/26 18:48:26 DAG Input file is diamond.dag
9/26 18:48:26 Rescue DAG will be written to diamond.dag.rescue
9/26 18:48:26 All DAG node user log files:
9/26 18:48:26   C:\condor\examples\dagman\diamond.log (Condor)
9/26 18:48:26 Parsing diamond.dag ...
9/26 18:48:26 Dag contains 4 total jobs
9/26 18:48:26 Deleting any older versions of log files...
9/26 18:48:26 Bootstrapping...
9/26 18:48:26 Number of pre-completed nodes: 0
9/26 18:48:26 Registering condor_event_timer...
9/26 18:48:27 Got node A from the ready queue
9/26 18:48:27 Submitting Condor Node A job(s)...
9/26 18:48:27 submitting: condor_submit -a dag_node_name' '=' 'A -a
+DAGManJobId' '=' '33 -a DAGManJobId' '=' '33 -a submit_event_notes'
'=' 'DAG' 'Node:' 'A -a +DAGParentNodeNames' '=' '"" A.submit
9/26 18:48:28 From submit: Submitting job(s). 9/26 18:48:28 From
submit: Logging submit event(s). 9/26 18:48:28 From submit: 1 job(s)
submitted to cluster 34. 9/26 18:48:28   assigned Condor ID (34.0)
9/26 18:48:28 Just submitted 1 job this cycle...
9/26 18:48:28 Event: ULOG_SUBMIT for Condor Node A (34.0)
```

```
9/26 18:48:28 Number of idle job procs: 1
9/26 18:48:28 Of 4 nodes total:
9/26 18:48:28  Done     Pre   Queued    Post   Ready   Un-Ready
Failed
9/26 18:48:28   ===      ===      ===      ===      ===         ===
===
9/26 18:48:28     0        0        1        0        0           3
0
9/26 18:53:33 Event: ULOG_EXECUTE for Condor Node A (34.0)
9/26 18:53:33 Number of idle job procs: 0
9/26 18:53:33 Event: ULOG_JOB_TERMINATED for Condor Node A (34.0)
9/26 18:53:33 Node A job proc (34.0) completed successfully.
9/26 18:53:33 Node A job completed
9/26 18:53:33 Number of idle job procs: 0
9/26 18:53:33 Of 4 nodes total:
9/26 18:53:33  Done     Pre   Queued    Post   Ready   Un-Ready
Failed
9/26 18:53:33   ===      ===      ===      ===      ===         ===
===
9/26 18:53:33     1        0        0        0        2           1
0
```

En este caso se puede observar como se ponen en la cola las cuatro tareas, y empieza con el lanzamiento de la primera tarea, en este caso la tarea A, que se añade al clúster 34, mostrándose que uno de los nodos está en la cola (Queued) y hay tres tareas hijas de la tarea A que no están listas para ser usadas (Un-Ready). Una vez que la tarea A se completa, las tareas B y C pasan a estas listas para ser completadas (Ready), quedando únicamente a la espera la tarea D (Un-Ready). En el array argv se muestra la localización de los archivos necesarios para llevar a cabo la ejecución de las tareas. Una manera más sencilla de comprobar el éxito o fracaso de una tarea es mediante los archivos .*err*, que a menos que ocurra algo, tendrán un tamaño nulo.

4.5 Obtención de resultados.

4.5.1 Tareas dagman

Cuando se termina la ejecución de la tarea pueden ocurrir dos cosas:

- Si la tarea termina con éxito, se obtendrá un archivo de salida por cada tarea que se desee, si así se ha especificado, o bien un único archivo de salida con el resultado final. En el caso de la tarea que hemos planteado se obtiene un archivo por cada tarea que se ha planteado: A.out, B.out, C.out, D.out.

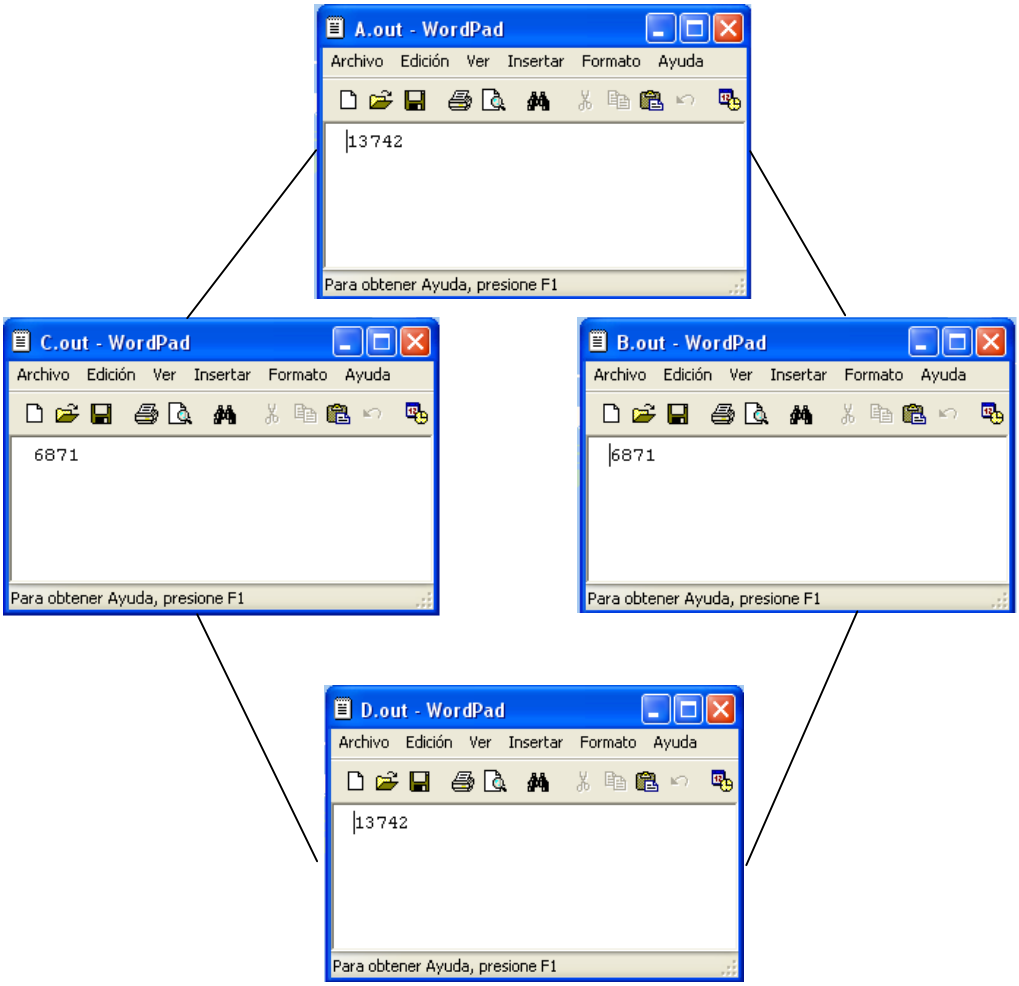


Figura 4. 7 Archivos de salida de las tareas Dagman

- Si hay algún problema el archivo dagman.out mostrará una salida parecida a la siguiente:

```
9/26 18:54:03 Got node D from the ready queue
9/26 18:54:03 Submitting Condor Node D job(s)...
9/26 18:54:03 submitting: condor_submit -a dag_node_name' '=' 'D -a
+DAGManJobId' '=' '33 -a DAGManJobId' '=' '33 -a submit_event_notes' '='
'DAG' 'Node:' 'D -a +DAGParentNodeNames' '=' '"B,C" D.submit
9/26 18:54:04 From submit: Submitting job(s). 9/26 18:54:04 From submit:
Logging submit event(s). 9/26 18:54:04 From submit: 1 job(s) submitted to
cluster 37. 9/26 18:54:04     assigned Condor ID (37.0)
9/26 18:54:04 Just submitted 1 job this cycle...
9/26 18:54:04 Event: ULOG_SUBMIT for Condor Node D (37.0)
9/26 18:54:04 Number of idle job procs: 1
9/26 18:54:04 Of 4 nodes total:
9/26 18:54:04  Done     Pre   Queued    Post   Ready   Un-Ready   Failed
9/26 18:54:04   ===     ===      ===     ===     ===         ===      ===
9/26 18:54:04     3       0        1       0       0           0        0
9/26 18:54:14 Event: ULOG_EXECUTE for Condor Node D (37.0)
9/26 18:54:14 Number of idle job procs: 0
9/26 18:54:14 Event: ULOG_JOB_TERMINATED for Condor Node D (37.0)
9/26 18:54:14 Node D job proc (37.0) failed with status 1.
9/26 18:54:14 Number of idle job procs: 0
9/26 18:54:14 Of 4 nodes total:
9/26 18:54:14  Done     Pre   Queued    Post   Ready   Un-Ready   Failed
```

```
9/26 18:54:14   ===      ===       ===      ===      ===         ===       ===
9/26 18:54:14     3        0         0        0        0           0         1
9/26 18:54:14 ERROR: the following job(s) failed:
9/26 18:54:14 ---------------------- Job ----------------------
9/26 18:54:14       Node Name: D
9/26 18:54:14          NodeID: 3
9/26 18:54:14     Node Status: STATUS_ERROR
9/26 18:54:14 Node return val: 1
9/26 18:54:14           Error: Job proc (37.0) failed with status 1
9/26 18:54:14 Job Submit File: D.submit
9/26 18:54:14   Condor Job ID: (37)
9/26 18:54:14       Q_PARENTS: 1, 2, <END>
9/26 18:54:14       Q_WAITING: <END>
9/26 18:54:14      Q_CHILDREN: <END>
9/26 18:54:14 ---------------------------------------<END>
9/26 18:54:14 Aborting DAG...
9/26 18:54:14 Writing Rescue DAG to diamond.dag.rescue...
9/26 18:54:14 Note: 0 total job deferrals because of -MaxJobs limit (0)
9/26 18:54:14 Note: 0 total job deferrals because of -MaxIdle limit (0)
9/26 18:54:14 Note: 0 total PRE script deferrals because of -MaxPre limit
(0)
9/26 18:54:14 Note: 0 total POST script deferrals because of -MaxPost
limit (0)
9/26 18:54:14 **** condor_scheduniv_exec.33.0 (condor_DAGMAN) EXITING
WITH STATUS 1
```

En el archivo se indica que se ha producido un error, indicando que la tarea ha finalizado con STATUS 1. Además se crea el archivo *.rescue*, que puede emplearse para intentar solventar los posibles problemas del grafo. En dicho archivo se muestra con el comando DONE aquellas tareas finalizadas con éxito, para que en la próxima ejecución sólo intenten ejecutarse las tareas que fallaron:

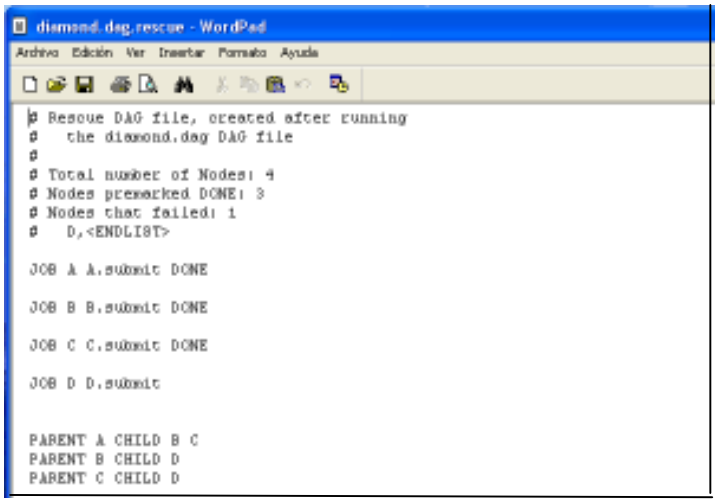


Figura 4. 8 Archivo *.rescue* de Dagman

Una vez arreglado el posible fallo habría que ejecutar lo siguiente:

[condor@maquina2 dagman]$ condor_submit_dag diamond.dag.rescue

De este modo solo intentará ejecutar aquellas tareas que fallaron.

4.5.2 Tareas Matlab

Como ya se explicó en el punto 4.3.2, no ha sido posible comprobar el funcionamiento de Matlab en Condor, pero si se hubiera conseguido, los comandos serían condor_q y condor_status, que fueron los que se usaron en las tareas Dagman, así como el archivo Matlab.log.

4.5.3 Tareas en C y Fortran

Para ver la evolución de la tarea propuesta son útiles los comandos condor_status y condor_q, como ya explicó en las tareas anteriores. Los resultados de los archivos son muy parecidos.

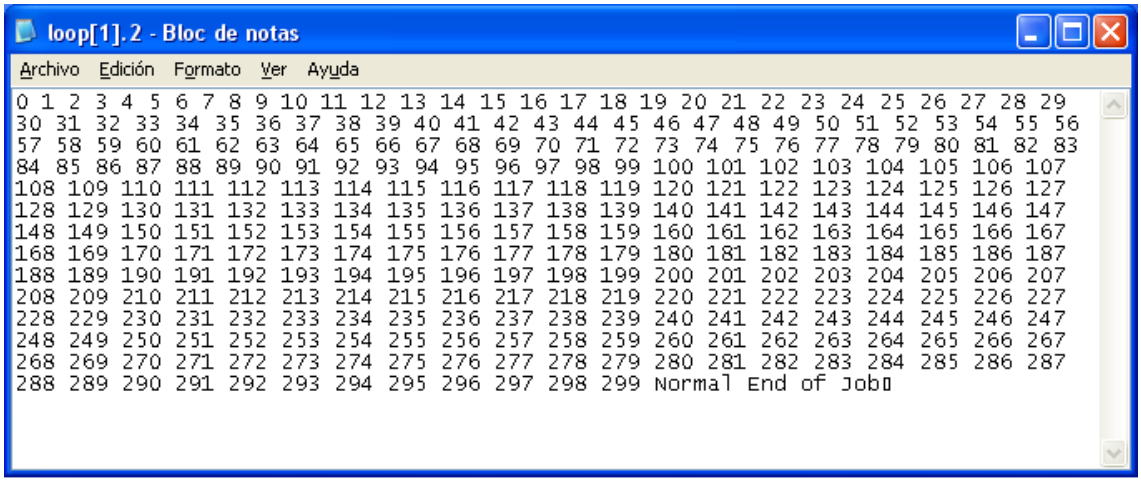


Figura 4. 9 Ejemplo de la salida de un archivo en C

Capítulo 5
Conclusiones y líneas futuras

5.1 Conclusiones

En el primer capítulo de este proyecto se plantearon diferentes objetivos a cumplir en este proyecto final de carrera. Dichos objetivos de idearon a partir de la idea de implementar un entorno de alto rendimiento computacional para aprovechar los recursos computacionales de los que dispone el Área de Ingeniería Telemática sin que ello supusiera un coste extra de hardware o de software y de forma totalmente transparente al usuario, es decir, el personal docente e investigador del área.

En este capitulo se analiza si dichos objetivos han sido cubiertos, extrayendo conclusiones sobre los distintos entornos computacionales existentes, la elección de Condor, el funcionamiento del mismo a nivel usuario y a nivel procesos, sistemas de ficheros, etc, el hardware y el software disponible y utilizado para llevar a cabo el proyecto, los lenguajes de programación usados y los resultados obtenidos tras la implementación del entorno computacional.

En el Capitulo 2 se hizo una valoración de los entornos de computación que hasta el momento existen. Entre los distintos entornos HPC, HTC y GRID se han ido analizando cada uno de ellos hasta llegar a la elección de Condor como entorno HTC que más convenía a las especificaciones de requisitos que se tenían a disposición. En dicho capitulo también se han descrito las características principales de Condor, la arquitectura y modo de funcionamiento así como los distintos “universos” en los que se puede trabajar.

En el Capítulo 3 se describen los pasos que se siguieron para la correcta instalación de un entorno Condor tanto en un S.O. Windows como en Linux así como la puesta en marcha de los procesos (demonios) del entorno. Hay que destacar que para Condor solo se puede instalar en los sistemas operativos indicados en su página. En un intento por trabajar bajo las condiciones de software impuestas en muchos de los laboratorios del área, se intentó instalar Condor-RedHat bajo un entorno Suse, que era el que había instalado en los PCs del laboratorio, lo que desembocó en varios problemas de compatibilidades en archivos de Condor. Además, también surgieron problemas en cuanto a los archivos de configuración de Condor, ya que la aplicación tiene varios archivos de configuración y líneas adicionales que se le pueden añadir para añadir funcionalidades a la aplicación.

En el capitulo 4, y previa comprobación de que el entorno de computación Condor funcionaba, se realizó un estudio sobre que tipo de tares se podían lanzar y cómo debían lanzarse, además de recopilar toda la información referente a que requisitos imponía el lanzamiento, ejecución y recopilación de datos obtenidos de una tarea.

Como se puede observar en los distintos capítulos que conforman este proyecto, se han cumplido satisfactoriamente los objetivos que se marcaron en la sección objetivos del capítulo de Introducción, quedando la aplicación a futuras ampliaciones que se enumeran en la siguiente sección.

5.2 Líneas futuras

Teniendo en cuenta que el proyecto se ha realizado sobre una red prototipo, la línea futura "inmediata" será la migración de lo realizado en la red prototipo a uno de los laboratorios docentes del Área de Ingeniería Telemática.

Tras esa migración, y utilizando la aplicación realizada en un proyecto paralelo a este, el personal docente e investigador podría acceder al Condor Pool creado en el laboratorio via web, sin tener que conocer y/o aprender el funcionamiento, los procesos o los comandos relativos a Condor.

Otra de las líneas futuras inminentes sería, una vez comprobado que en el laboratorio todo funciona correctamente, instalar tantos Condor Pools como laboratorios docentes tiene el Área de Ingeniería Telemática, creando así, por medio del mecanismo Flocking explicado en el apéndice de este proyecto un Condor Grid. De esta forma, a la hora de lanzar las tareas, los usuarios podrían decidir el Condor Pool donde desean lanzarlas, o la arquitectura de las máquinas donde deben ejecutarse esas tareas, etc.

Siguiendo con la expansión de Condor Grid y el aprovechamiento de los recursos desaprovechados de la Escuela de Telecomunicaciones, podría crearse un gran Condor GRID en la escuela, utilizando, no solo los laboratorios del AIT sino también el resto de laboratorios que forman parte de la escuela. Este aumento de la capacidad de procesamiento podría llegar hasta el orden de TFLOPS TFLOPS (10^{12} FLOPS). De esta forma, no solo se aprovecharían al máximo los recursos de la escuela sino que también se diseñaría un GRID con diversidad de Condor Pools, cada uno implementado con un universo distintos, diversidad de sistemas operativos, requisitos de memoria, prioridades en los laboratorios, etc.

Apéndice A. Conexión de laboratorios mediante el mecanismo Flocking.

Para finalizar con los objetivos que se fijaron al principio del proyecto se hace necesario el uso de un mecanismo que permita la comunicación entre los diferentes laboratorios del área de Ingeniería Telemática, con el fin de que, al lanzar una tarea, sea posible usar el total del poder de procesamiento que se tiene en los laboratorios del área y cuyos resultados se presentaron en la introducción de este proyecto final de carrera.

Si en cada laboratorio se supone que hay configurado un Condor Pool, habrá al menos un Central Manager, por ejemplo el servidor de cada laboratorio, y mediante la herramienta de Flocking se puede crear un entorno Grid que permita la comunicación entre los diferentes laboratorios. El éxito de Grid reside en la utilización de recursos que se encuentran dentro de diferentes dominios administrativos. Debido a la rápida evolución de los competidores, Condor posee su propio mecanismo nativo, para realizar cálculos Grid, así como para llevar a cabo interacciones con otros sistemas Grid.

Flocking es un mecanismo nativo que permite que las tareas lanzadas en Condor desde un Pool, sea ejecutada en otro Pool diferente. Flocking debe de ser activado a través del archivo de configuración en cada uno de los Pool, que en este caso es cada uno de los laboratorios del área de Telemática. Una ventaja de Flocking es que las tareas migran desde un Pool a otro basándose en la capacidad que posean las máquinas del Pool para ejecutar las tareas. Cuando el Pool local no está capacitado para ejecutar la tarea (porque por ejemplo todas las máquinas estén incapacitadas u ocupadas), la tarea es enviada a otro Pool. Otra ventaja del uso de Flocking es que el usuario no es consciente de ello, puesto que el fichero de descripción submit es independiente del mecanismo de flocking.

Otras formas de cálculo usando Grid son habilitadas mediante el uso del universo Grid, y más aún mediante la especificación grid_type. Para cualquier tarea de Condor, la tarea es lanzada en una máquina del Pool local de Condor. La localización donde se ejecuta es identificada como máquina remota, o como recurso remoto. Todos estos mecanismos Grid que Condor ofrece, se distinguen por el software ejecutado en el recurso remoto.

Conectando los Pools de Condor mediante Flocking

Flocking es la manera en la que Condor permite que las tareas que no pueden ejecutarse inmediatamente en el Pool donde fue lanzada, puedan ejecutarse en un Pool diferente. Las variables del fichero de configuración permiten que el demonio condor_schedd implemente Flocking.

Configurar Flocking

La configuración de Flocking es bastante sencilla, utilizando pocas variables de configuración. Si se desea que las tareas de una máquina A, puedan ser ejecutadas en otro Pool B, entonces en la configuración hay que tener en cuenta las siguientes variables:

FLOCK_TO

Posee una lista separada por comas de los Central Manager de los Pools, a donde las tareas de la máquina A pueden ir para ser ejecutadas.

FLOCK_COLLECTOR_HOSTS

Es la lista de los demonios condor_collector dentro de los Pools que las tareas de la máquina A pueden mandar. En muchas ocasiones, es la misma que FLOCK_TO y puede ser definida de la siguiente manera:

FLOCK_COLLECTOR_HOSTS= $(FLOCK_TO)

FLOCK_NEGOTIATOR_HOSTS

Es la lista de los demonios condor_negotiator dentro de los Pools que las tareas de la máquina A pueden migrar. En muchos casos es la misma que la que se define en FLOCK_TO y se puede definir como:

FLOCK_NEGOTIATOR_HOSTS= $(FLOCK_TO)

HOSTALLOW_NEGOTIATOR_SCHEDD

Provee un nivel de acceso basado en host y una lista de autorización para el demonio condor_schedd que permite la negociación (por razones de seguridad) desde la máquina A hacia las máquinas de los Pools donde las tareas de la máquina A pueden migrar. Su valor suele ser el que se da por defecto:

HOSTALLOW_NEGOTIATOR_SCHEDD=$(COLLECTOR_HOST),
$(FLOCK_NEGOTIATOR_HOST)

Las macros de configuración que deben de usarse en el Pool de destino B son las que autorizan a las tareas de la máquina A migrar hacia el Pool B. Estas macros, no son más que una lista de máquinas separadas por comas (FLOCK_FROM), hacia donde las tareas pueden migrar.

HOSTALLOW_WRITE_COLLECTOR = $(HOSTALLOW_WRITE), $(FLOCK_FROM)

HOSTALLOW_ WRITE_STARTD = $(HOSTALLOW_WRITE), $(FLOCK_FROM)

HOSTALLOW_READ _COLLECTOR = $(HOSTALLOW_WRITE), $(FLOCK_FROM)

HOSTALLOW_READ_STARTD = $(HOSTALLOW_WRITE), $(FLOCK_FROM)

Caso particular de los laboratorios de la universidad

En el caso particular de la universidad hay nueve laboratorios, teniendo todos salvo el I+D+I-1 y el I+D+I-3, su propio servidor. Por ello el primer paso consistiría en la instalación de un servidor en cada uno de los laboratorios, para de esa manera poder emplearlos.

El siguiente paso sería la asignación de una IP estática a cada uno de los servidores de los laboratorios que actuarán como Central Manager como por ejemplo los que se muestran en la siguiente tabla:

IT-1	IT-2	IT-3	IT-4	IT-5	IT-6	I+D+I-1	I+D+I-2	I+D+I-3
192.168.1.1	192.168.1.2	192.168.1.3	192.168.1.4	192.168.1.5	192.168.1.6	192.168.1.7	192.168.1.8	192.168.1.9

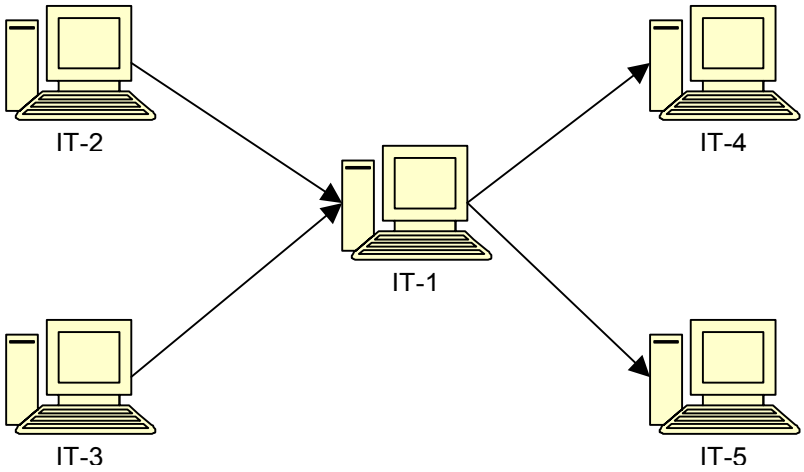
Apéndice 1. 1 Asignación IP estática en servidores AIT

Hay que indicar que estos servidores disponen de una IP estática pública, que también podría utilizarse para tales efectos.

El siguiente paso sería indicar al Central Manager de cada laboratorio hacia donde permite hacer el mecanismo de Flocking, y hacia donde permite realizar Flocking. Por ejemplo, si se quisiera que el laboratorio IT-1 recibiera tareas del IT-2 y el IT-3, y enviara tareas al IT-5 y al IT-6, teniendo sólo el Central Manager del IT-1 los permisos de administrador, y dejando que todas las máquinas tuvieran permisos de lectura y escritura, la parte de Flocking del archivo de configuración global sería:

```
## Flocking: Submitting jobs to more than one pool
## Flocking allows you to run your jobs in other pools, or lets
## others run jobs in your pool.
## To let others flock to you, define FLOCK_FROM.
## To flock to others, define FLOCK_TO.
FLOCK_FROM = 192.168.1.2, 192.168.1.3
FLOCK_TO = 192.168.1.4, 192.168.1.5
FLOCK_NEGOTIATOR_HOSTS = $(FLOCK_TO)
FLOCK_COLLECTOR_HOSTS = $(FLOCK_TO)
ºHOSTALLOW_ADMINISTRATOR = $(FULL_HOSTNAME)
HOSTALLOW_OWNER = $(FULL_HOSTNAME), $(HOSTALLOW_ADMINISTRATOR)
HOSTALLOW_READ = *
HOSTALLOW_WRITE = *
## Negotiator access.  Machines listed here are trusted central
## managers.  You should normally not have to change this.
HOSTALLOW_NEGOTIATOR = $(CONDOR_HOST)
## Now, with flocking we need to let the SCHEDD trust the other
## negotiators we are flocking with as well.  You should normally
## not have to change this either.
HOSTALLOW_NEGOTIATOR_SCHEDD = $(CONDOR_HOST), $(FLOCK_NEGOTIATOR_HOSTS)
## Flocking Configs.  These are the real things that Condor looks at,
## but we set them from the FLOCK_FROM/TO macros above.  It is safe
## to leave these unchanged.
HOSTALLOW_WRITE_COLLECTOR = $(HOSTALLOW_WRITE), $(FLOCK_FROM)
HOSTALLOW_WRITE_STARTD    = $(HOSTALLOW_WRITE), $(FLOCK_FROM)
HOSTALLOW_READ_COLLECTOR  = $(HOSTALLOW_READ), $(FLOCK_FROM)
HOSTALLOW_READ_STARTD     = $(HOSTALLOW_READ), $(FLOCK_FROM)
```

El esquema de conexión de la red de laboratorios creada sería la siguiente:



Apéndice 1. 2 Ejemplo de red creada mediante Flocking

Por ello, para realizar una conexión completa de todos los laboratorios entre sí, sólo sería necesario en FLOCK_FROM y en FLOCK_TO, escribir una lista completa de todos los Central Manager, dejando que cada Central Manager de manera individual manejara las cuestiones administrativas.

Bibliografía

La información utilizada en este proyecto, se ha obtenido de diferentes páginas Web y documentos obtenidos de las páginas oficiales de los entornos de computación y páginas donde hacen referencias a dichos entornos, como ha continuación se redactan:

Paginas Oficiales de los entornos de computación:

- OpenMosix: http://openmosix.sourceforge.net/
- Beowulf: http://www.beowulf.org/overview/index.html
- Ibis: http://www.cs.vu.nl/ibis/
- Globus Toolkit: http://www.globus.org/toolkit/
- Vgrid: http://vgrid.sourceforge.net/index.html
- OurGrid: http://www.ourgrid.org/
- Condor: http://www.cs.wisc.edu/condor/

Paginas de información general:

- Centro informático científico de Andalucía: http://www.cica.es/herramientas-cientificas-open-source.html
- Wikipedia (GRID): http://es.wikipedia.org/wiki/Grid
- Wikipedia (CLUSTER): http://es.wikipedia.org/wiki/Cluster_de_computadores
- Wikipedia (Superordenador): http://es.wikipedia.org/wiki/Supercomputadora

Libros de interés:

- Ian Foster, Carl Kesselman (1999). La Rejilla: libro azul para una nueva infraestructura informática (The Grid: Blueprint for a New Computing Infrastructure). Morgan Kaufmann Publishers. ISBN.
- Fran Berman, Anthony J.G. Hey, Geoffrey Fox (2003). La rejilla informática: haciendo realidad la Infraestructura Global (Grid Computing: Making The Global Infrastructure a Reality). Wiley. ISBN.