Sistemas Informáticos
Curso 2003-04

Estudio de un Planificador de recursos Software Distribuidos y ampliación a la Gestión Distribuida de Hardware

Ana María Vélez Villanueva
Germán Hurtado Martín
Ignacio José Horrillo Sancho

Dirigido por:
Prof. Julio Septién del Castillo
Dpto. de Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense de Madrid

ÍNDICE

RESUMEN DEL PROYECTO EN ESPAÑOL

El objetivo de este proyecto, relacionado con un proyecto mayor de investigación, el GHADIR (Gestión de Hardware Dinámicamente Reconfigurable), era instalar y analizar un planificador de recursos software distribuidos bien conocido en el mundo académico, Condor, y estudiar las posibilidades de ampliación a la gestión de recursos de hardware reconfigurable distribuidos. Si esto era factible se implementaría dicha ampliación; en caso contrario, se implementaría un planificador simplificado que gestione el nuevo recurso.

Tras estudiar Condor y su funcionamiento se vió que era posible utilizarlo para los fines especificados, gestionar FPGAs que se encuentran distribuidas en una red, así que se procedió a llevar a cabo las modificaciones necesarias e implementar los mecanismos necesarios para ello.

El presente documento es pues tanto una introducción a las posibles configuraciones de una FPGA y al sistema Condor como un pequeño estudio sobre dicho sistema y sobre los pasos a seguir para utilizarlo para gestionar recursos de hardware reconfigurable distribuidos.

PROJECT SUMMARY IN ENGLISH

The goal of this project, related with a larger investigation project, the GHADIR (Gestión de Hardware Dinámicamente Reconfigurable, dinamically reconfigurable hardware management), was installing and analyzing a well-known in the academic community distributed software resources scheduler, Condor, and studying the possibilities of extending it in order to manage distributed reconfigurable hardware resources. If this was feasible, that extension would be implemented; otherwise, a simplified scheduler for managing the new resource would be implemented.

After studying Condor and its performance it was seen that using it for the specified goals, managing FPGAs which are distributed in a network, was possible, so we proceeded to perform the necessary modifications and to implement the necessary mechanisms for it.

The present document is then as much as an introduction to the possible configurations of a FPGA and to the Condor system as a little study on this system and on the steps which one must follow to use it in order to manage distributed reconfigurable hardware resources.

LISTA DE PALABRAS CLAVE / KEYWORDS LIST

Condor
Hardware Dinámicamente Reconfigurable
Planificador de Tareas
GHADIR
FPGA
ClassAd
GRID

LICENCIA

I.J. Horrillo Sancho G. Hurtado Martín A.M. Vélez Villanueva

1. MARCO DEL PROYECTO

1.1. INTRODUCCIÓN

Hay recursos hardware reconfigurables que a menudo están infrautilizados a pesar de que son recursos caros. Por tanto es conveniente dar mayor uso a este tipo de recursos y permitir que se puedan compartir entre varios usuarios, obteniendo así un mayor aprovechamiento y permitiendo de este modo tanto su utilización por parte de distintos usuarios como la aceleración de cálculos complejos que se podrían ejecutar de forma distribuida entre varios recursos. En este proyecto lo que se ha intentado ha sido el conseguir compartir uno de estos tipos de recursos, las tarjetas FPGA, englobado en un proyecto mayor de hardware dinámicamente reconfigurable en el que trabaja la Facultad de Informática de la Universidad Complutense de Madrid.

1.2. PROYECTO DE INVESTIGACIÓN GHADIR

El GHADIR (Gestión de Hardware Dinámicamente Reconfigurable) es un proyecto de investigación de la Facultad de Informática de la Universidad Complutense de Madrid que tiene como objetivo el estudio de los problemas que presenta la gestión de recursos HW dinámicamente reconfigurables en sistemas de propósito general que ejecutan una secuencia de tareas HW aleatorias. Obviamente las FPGA's son idóneas para este objetivo, y sobre ellas se centrará nuestro proyecto. Así pues, habrá que gestionar la FPGA para planificar las tareas, cargarlas, asignarles recursos, etc., pero también habrá que encargarse de la cuestión que planteábamos al principio: poder hacer que la FPGA esté disponible para otras máquinas aparte de la suya, y por tanto tendremos que ocuparnos de la gestión del recurso hardware distribuido a este nivel. (Para más información véase [1])

1.3. PROYECTO DE SISTEMAS INFORMÁTICOS

Es de esta segunda parte de la que tratará este proyecto, gestionar el uso de una FPGA accesible desde distintos puestos, de modo que no se limite a ser utilizada por la misma máquina y poder así alcanzar un uso óptimo como se planteó anteriormente. Para ello, dado que en los últimos años se ha puesto un especial interés en los sistemas de gestión de tareas software para computación distribuida, se ha pensado en estudiar el uso de uno de estos sistemas para estos fines (puesto que el objetivo de estos sistemas es también aprovechar los recursos software con los que cuenta un grupo de máquinas conectadas), viendo si es posible, y si merece la pena, ampliarlo convenientemente para adaptarlo a la gestión de hardware reconfigurable.

2. PLANTEAMIENTO DEL PROBLEMA

Había que encargarse de la gestión de las distintas tareas que debían llevar a cabo las FPGA's y la forma en la que eran enviadas a éstas, aparte de, claro está, hacer que en todo momento estuviese disponible la información de los recursos de cada máquina, y que las demás máquinas pudieran tener acceso a dicha información. Para ello se barajaron dos opciones: hacer un sistema específico para gestionar estos recursos, o bien aprovechar alguno de los sistemas de gestión de tareas (JMS, Job Management System) ya existentes, intentando, como se dijo antes, ampliarlo convenientemente para su uso con las FPGA's.

La primera opción suponía partir de cero, pero con la ventaja de que sería un sistema diseñado específicamente para nuestros intereses, con lo cual sabemos de antemano que se ajustará perfectamente a lo que queremos, esto es, podrá hacer especial hincapié en los recursos en los que deseamos fijarnos, y dejar a un lado aquellos otros que no nos interesen particularmente. Esto conllevará el hecho de que por lo tanto será un sistema no demasiado "pesado", al menos si nos limitamos a ocuparnos sólo de ciertos recursos, ya que tendrá las funcionalidades justas que deseamos que estén disponibles. Además, de este modo podremos, incluso, dejar la puerta abierta a poder hacer futuras ampliaciones de una manera sencilla, ya que al ser nosotros los creadores podemos tener en cuenta a la hora de diseñarlo algunas cosas que se podrían querer hacer en el futuro. Por otro lado, sabemos que el programa podrá hacer todo lo que queramos sin encontrarnos con ninguna restricción, ya que por ejemplo si usamos otro programa quizás nos topemos con restricciones (e inconvenientes) tales como "no funciona en el sistema operativo X" o "no se puede hacer la cosa Y". Aparte de todo esto, los JMS disponibles son sistemas con bastantes opciones y por tanto bastante grandes en comparación con lo que sería nuestro sistema dedicado especialmente a la gestión de las FPGA's, lo que implica que si queremos extenderlo para el objetivo que nos hemos planteado primero tendremos que saber cómo utilizarlo y, después, cómo funciona exactamente, para de este modo poder llevar a cabo nuestro trabajo de forma efectiva, lo que nos llevaría una buena cantidad de tiempo, pues por muy bien documentado que esté un trabajo es imposible continuar el trabajo de otras personas poniéndose manos a la obra de forma inmediata.

Es obvio que todo tiene sus inconvenientes, y aquí el principal es, como puede suponerse, que hay que partir de cero mientras que hay algunos JMS de dominio público que quizás pudieran ser ampliados para su uso con hardware reconfigurable, lo cual ahorraría tiempo y trabajo, y además proporcionaría a nuestro sistema una mayor "potencia" puesto que dichos programas, al tener ya otras funcionalidades (de hecho la que nos interesa, para el hardware reconfigurable, no la tienen, es precisamente lo que hemos de añadir) tienen un mayor número de usos que quizás sí que nos fuesen útiles. Además, al ampliar se podría llegar a dar con un sistema con múltiples usos y, al estar hablando de sistemas más o menos conocidos y extendidos, el trabajo puede resultarle útil a más gente.

En cualquier caso, esta primera opción, aunque estaba presente, era secundaria y sólo surgió como opción a considerar seriamente en el curso del desarrollo del proyecto, ya que como se comentó en el marco del proyecto la idea principal de esta parte nuestra del proyecto consistiría en ampliar uno de estos JMS, o estudiar las posibilidades que había de ampliarlo, si era factible o no, y si merecía la pena. Así que a pesar de las ventajas de la primera opción de la que acabamos de hablar, nos encaminamos directamente hacia la segunda opción: ampliar un JMS ya existente.

El primer problema en este caso era cuál de los JMS disponibles elegir, y por razones que explicaremos en el [Apartado 4], se optó finalmente por Condor, un sistema de dominio público creado en la Universidad de Wisconsin-Madison, e inicialmente lo usaríamos bajo el sistema operativo Linux, dado que aunque el equipo de desarrolladores de Condor había sacado ya una versión para Windows, al principio, desde las primeras versiones, sólo funcionaba en Linux y por tanto era un sistema en donde se habían hecho más pruebas y donde, en definitiva, estaba más rodado. De todos modos posteriormente probaríamos también la versión de Windows y de hecho el proyecto se ha realizado para que funcione en ambos sistemas.

Aparte de las ventajas que nos puede proporcionar Condor en concreto, las ventajas de trabajar a partir de un sistema ya existente son varias. No hay que empezar desde cero, con lo cual nos saltamos prácticamente toda la parte del diseño (decimos "prácticamente" ya que sí hay que diseñar la ampliación), trabajamos con un sistema que ha sido probado, y nos ofrece múltiples funcionalidades que, si bien seguramente no son necesarias para lo que queremos hacer, es posible que sean útiles para futuros usos y por tanto si entonces queremos hacer

uso de una de estas utilidades podamos usar únicamente Cóndor para gestionar tanto hardware reconfigurable como recursos software, en vez de tener que echar mano de varios programas, que siempre suele ser más engorroso.

De todos modos está claro que estas ventajas llevan asociadas algunos inconvenientes. Por ejemplo, si queremos hacer uso únicamente de la gestión de las FPGA's, estaremos usando un programa demasiado potente, y pesado, para una tarea que se podría solucionar de modo más sencillo con un programa dedicado únicamente a gestionar dicho recurso, ya que obviamente este programa no ha sido diseñado para nuestro objetivo. Esto también implica que quizás deseemos, o necesitemos, hacer ciertas cosas que el sistema no permite (como ya mencionamos antes), lo cual sería un gran inconveniente ya que habría que ingeniárselas para saltar de algún modo esa barrera perdiendo en ello un tiempo que quizás se haya malgastado de cualquier manera (y en mayor cantidad) si finalmente descubrimos que no hay forma alguna de hacerlo y que llevar a cabo nuestros objetivos con dicho JMS es prácticamente imposible. Además, también se ha comentado, estamos continuando el trabajo de otras personas y, pese a la información que puede venir suministrada tanto como con el propio JMS como la que se puede encontrar acerca de él ya sea en Internet, en artículos, o preguntando a los propios desarrolladores, continuar un trabajo ajeno conlleva pasar un cierto tiempo aprendiendo el uso y funcionamiento del programa.

Como se puede ver, las ventajas de una opción son las desventajas de la otra y viceversa, así que realmente ninguna de las dos opciones es mejor que la otra, y dado que la idea era ampliar un JMS, esa es la opción que escogimos, ya que a fin de cuentas, era tan válida como la de empezar desde cero. De no haber sido así, y a pesar de la idea inicial de ampliar un JMS ya existente, tendríamos que haber escogido a la fuerza la primera opción, ya fuese al principio del proyecto o, si hubiésemos calculado mal, a mitad del mismo. Afortunadamente no fue así y la opción del JMS pudo seguir adelante.

Por tanto el planteamiento inicial era el siguiente: instalar Condor en Linux y probarlo para uso con recursos software; una vez comprendido su funcionamiento, haríamos las modificaciones necesarias para conseguir que funcionase con las FPGA's (esto es, que fuese capaz de administrar ese tipo de recursos hardware). Finalmente, si fuese posible, intentaríamos conectarlo con el resto del proyecto, o sea, conectarlo con un programa que fuese quien manejase realmente las FPGA's, puesto que nuestra parte sólo se encargaría de la gestión a nivel de la red de ordenadores.

3. DISTINTAS FORMAS DE GESTIÓN DE HARDWARE RECONFIGURABLE

3.1. INTRODUCCIÓN

El Hardware Reconfigurable es un recurso muy difundido y utilizado en entornos universitarios, ya sea para investigación o para practicas de los alumnos, también esta ampliamente implantado en ámbitos de diseño de nuevos componentes hardware, ya que proporciona una manera de testeo del hardware sin coste de fabricación, y además reutilizable. Este recurso consta generalmente de una o varias tarjetas con dispositivos Field-Programmable Gate Array (FPGA), que pueden permitir o no diferentes tipos de uso, ya sea desde el punto de vista de la reprogramación, que puede ser total o parcial, o de la forma de reprogramarlos, ya sea en tiempo real (Run-Time) o parando todas las tareas en proceso, para añadir una nueva tarea y volver a reprogramar nuevamente la FPGA.

En los últimos años, el tamaño y densidad de las FPGAs ha crecido, y han aparecido interesantes características como la posibilidad de reconfigurar dinámicamente parte de la FPGA con una nueva funcionalidad, sin tener que parar el resto de aplicaciones o tareas en curso (Run-Time), así como la posibilidad de gestionar este tipo de recursos en Una Dimensión (1D) o en Dos Dimensiones (2D). Estas características, que se resumen en [2] han despertado un gran interés en incorporar al Sistema Operativo (SO) la gestión del hardware dinámicamente reconfigurable (HwDR), dado que el tamaño actual de este tipo de dispositivos permite considerar tanto la posibilidad de múltiples tareas HW ejecutándose simultáneamente en una FPGA como la reconfiguración dinámica de la misma para realizar diversas tareas a lo largo del tiempo. De este modo un recurso reconfigurable actual (FPGA) puede verse como un área de proceso en una o dos dimensiones, capaz de ejecutar un conjunto de tareas de modo concurrente. Hay que tener en cuenta que cada una de estas tareas ha sido previamente compilada en un código reubicable que puede cargarse para su ejecución en una sección libre de la FPGA, de tal forma que cada tarea HW puede terminar o introducirse tareas nuevas en la FPGA, sin afectar al resto de tareas en ejecución.

Como se muestra en [3], el modo en que se gestione el espacio libre y se ubiquen las nuevas tareas, tendrá una influencia decisiva en el comportamiento de estos dispositivos, ya que al igual que en la memoria de un computador se produce el efecto negativo de la fragmentación, tanto interna como externa, el cual reduce el rendimiento del computador. En las FPGA se produce este mismo efecto, reduciéndose igualmente el rendimiento de la FPGA, por ello puede considerarse la posibilidad de reubicación de las tareas existentes en la FPGA (Desfragmentación), si la fragmentación crece demasiado, esto solo es necesario en algunos tipos de gestión de FPGA, ya que no todas las distintas formas de gestión producen este efecto, pero es destacable tenerlo en cuenta a la hora de planificar el uso de nuestra FPGA, para poder aprovecharla al máximo.

3.2. TIPOS DE GESTIÓN DE HARDWARE DINÁMICAMENTE RECONFIGURABLE

Los diferentes tipos de gestión de este tipo de hardware, se basan fundamentalmente en dos características que han de soportar las FPGAs, la posibilidad de reconfiguración dinámica, y la posibilidad de utilizarla en 1D o 2D. En función de estos requisitos, las diferentes formas de gestión son:

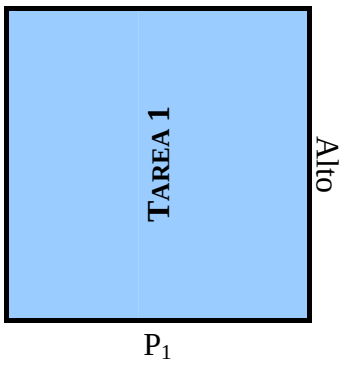
1. Gestión Estática
2. Gestión Dinámica
    - 2.1. Una Dimensión
        - 2.1.1. Celdas de tamaño uniforme y fijo
        - 2.1.2. Celdas de tamaño no uniforme y fijo
        - 2.1.3. Celdas de tamaño variable
    - 2.2. Dos Dimensiones
        - 2.2.1. Celdas de tamaño uniforme y fijo
        - 2.2.2. Celdas de tamaño no uniforme y fijo
        - 2.2.3. Celdas de tamaño variable

Paso ahora a definir en profundidad cada uno de estos tipos de gestión.

3.2.1. GESTIÓN ESTÁTICA

Este tipo de gestión, es la más sencilla de implementar y desarrollar, ya que la FPGA no necesita ser extremadamente potente pues no necesitamos ningún tipo de reconfiguración parcial (implementada en hardware), ni de divisiones en celdas básicas por parte del espacio utilizable de la FPGA, ya que está se utiliza como un todo, es decir el conjunto de tareas que se estén ejecutando han de estar unidas en un único bitmap, que las representa para poder así cargarlo sobre la FPGA, si se precisa de multitarea hardware, esta se implementara mediante alguna técnica software que cree un bitmap a partir de varios, teniendo en cuenta la extensión real de las tareas que representan ambos bitmaps.

Las ventajas de este tipo de gestión son, que el gasto en la infraestructura, para el uso de este tipo de sistemas, será sensiblemente inferior, a parte, la utilización y el modo de uso es el más rápido posible, pues no precisa calcular ningún dato adicional para establecer la FPGA que ejecutará nuestra tarea, ya que la FPGA solo dispone de dos estados, libre u ocupada, esto da mucha rapidez a la hora de planificar el uso de la FPGA. Finalmente la mayor de las ventajas es la ausencia de fragmentación, ya que al no dividirse el espacio, no se produce este fenómeno.



Sin embargo, las desventajas de está gestión son, que no se aprovecha totalmente la FPGA, y estos recursos que son de un coste elevado se han de intentar rentabilizar lo máximo posible, y que no permite introducir tareas nuevas si no es parando las actuales, ya que hay que generar un bitmap nuevo (que represente la unión de las tareas).

3.2.2. GESTIÓN DINÁMICA

Este tipo de gestión es la mas apropiada para el aprovechamiento máximo de estos dispositivos, pues permiten el uso casi total de la FPGA, también es verdad que para poder implementar esta gestión son necesarias unas FPGAs más potentes que permitan reconfiguración parcial y si es posible una distribución en 2D.

Los diferentes tipos de gestión dinámica dependen de, el tipo de celdas básicas que podamos definir, ya sean en 1D o 2D y del tipo de particiones que queramos definir, es decir particiones de un tamaño fijo, utilizando 1 o más celdas básicas o particiones de un tamaño variable, es decir que se nos permita, en tiempo de planificación, decidir el numero de celdas básicas a usar y su distribución en el espacio utilizable de la FPGA.

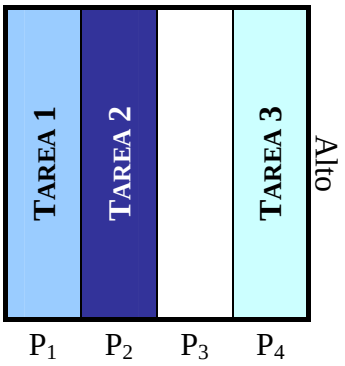
3.2.2.1. Una Dimensión

Para poder implementar esta gestión, es necesario dividir la FPGA en celdas básicas de una dimensión, es decir que su altura sea igual al alto de la FPGA (en pixeles o transistores) y el ancho un valor adecuado en función del número de divisiones deseadas, en resumen se divide la FPGA en un número finito de columnas.

3.2.2.1.1. Celdas de tamaño uniforme y fijo

Consiste en definir un número (N) de particiones del mismo tamaño (ancho) que ocupen todo el ancho de la FPGA, de esta manera podremos tener al mismo tiempo en ejecución N tareas, siempre y cuando cada una de ellas ocupe menos del tamaño de una partición y tenga la misma forma (en este caso columnas). Su implementación consistiría en llevar un entero con las particiones ocupadas o libres.

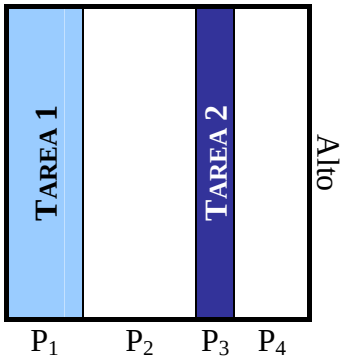
Este tipo de gestión tiene como ventajas el mayor aprovechamiento de la FPGA y la posibilidad de multitarea hardware real y que no se produce fragmentación externa.



Aunque como inconvenientes tiene, que si se produce fragmentación interna, que se da en todos los tipos de gestión, ya que siempre existe un tamaña de celda básica o partición, y aunque este tipo de fragmentación no es muy importante en algunos casos, en este si pues depende directamente del tamaño de la partición que decidimos, que ha de ser elegido de una forma muy precisa para así aprovechar la FPGA, ya que si es demasiado pequeño, no se podrán cargar la mayoría de tarea y si es demasiado grande se desperdicia mucho espacio (aumenta la fragmentación interna), el otro inconveniente fundamental es la imposibilidad de admitir tareas que no tengan forma de columna, que no siempre es posible conseguir.

3.2.2.1.2. Celdas de tamaño no uniforme y fijo

Este tipo de gestión pretende evitar la excesiva importancia que tiene el tamaño de la partición a la hora de aprovechar el recurso, de esta manera se pretenden definir un número finito de divisiones de la FPGA, en columnas, pero no del mismo ancho, con lo cual se pueden aprovechar al máximo dependiendo del tamaño de la tarea. Su implementación se basaría en llevar un array en el cual actuará como índice el tipo de partición y el contenido representaría el número de particiones de este tipo libres u ocupadas, el tipo estaría definido por el ancho de la partición.

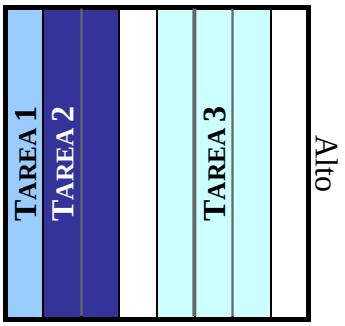


Las principales ventajes son, la casi ausencia de fragmentación interna, sumada a las anteriormente mencionadas en el apartado anterior.

Como desventajas, está el posible desaprovechamiento del área de la FPGA, al no poder dividir, en ocasiones, las particiones para poder aprovecharla completamente, si no hay particiones libre de un tamaño inferior, y al igual que antes la imposibilidad de admitir tareas que no tengan forma de columna.

3.2.2.1.3. Celdas de tamaño variable

Esta gestión reparte el espacio de la FPGA en tantas columnas como pixeles tiene de ancho la FPGA, creando así un conjunto de columnas básicas, las cuales se permiten unir en función de las necesidades de la tarea entrante, de esta forma se utilizan exactamente el número necesario de columnas básicas que ocupe la tarea a ejecutar. La implementación es la mas costosa de 1D, pues se deberían llevar dos lista dinámicas que representen las particiones ocupadas y los huecos, de tal forma que queden definidas en función de los índices de comienzo y final de cada partición, estos índices se representarían en función de las columnas básicas, y el contenido de cada eslabón de la lista contendría la información necesaria para la gestión, (tamaño, inicio, fin,…).



Respecto a las formas de gestión en una dimensión, está es la mejor posible en cuanto al aprovechamiento de la FPGA, sin embargo es la más costosa de implementar, pues es necesario llevar listas dinámicas de particiones y se genera mucha fragmentación externa, aunque no interna, ya que las columnas básicas son lo suficientemente pequeñas, para no desaprovechar nada de espacio.

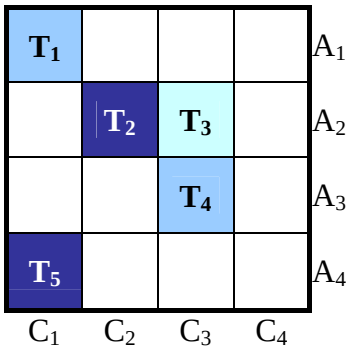
Aun así se sigue manteniendo el problema en cuanto a la forma de las tareas, las cuales han de ocupar todo el alto de la FPGA, para no desperdiciar superficie.

3.2.2.2. Dos Dimensiones

Para solucionar el problema geométrico de las tareas, es necesario usar FPGAs que permitan una gestión en 2D, de esta manera y aunque tiene una implementación más costosa, se pude aprovechar realmente las capacidades de la FPGA, sin embargo las tareas no pueden tener cualquier forma geométrica ya que el algoritmo para llevar la gestión seria costosísimo, no obstante, esta gestión se puede plantear con simplificaciones que ayudan mucho a la hora de rentabilizar el uso de estos recursos.

3.2.2.2.1. Celdas de tamaño uniforme y fijo

Es la implementación mas simple, simplemente se divide el área en particiones del mismo tamaño, en anchura y altura, y así basta con conocer cuantas están libres u ocupadas, ya que toda tarea entra indiferentemente en una partición u otra. La implementación es sencillamente un entero que represente el número de particiones libres u ocupadas.

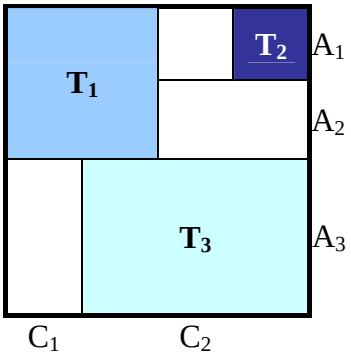


Las ventajas de este tipo de gestión son la facilidad de implementación y la posibilidad de incluir tareas con geometrías no alargadas, como era el caso de las columnas.

Y las desventajas son básicamente el desperdicio de espacio dentro de cada una de las particiones, ya que no es posible adaptar la partición lo mas posible a la tarea, lo que plantea también fragmentación interna, si no se escoge el tamaño adecuado de partición.

3.2.2.2.2. Celdas de tamaño no uniforme y fijo

Para dar mayores posibilidades a las tareas, se pueden definir un número finito de particiones, con geometrías diferentes, pensadas para albergar distintos tipos de tareas, esto amplia enormemente las posibilidades de uso de la FPGA. Su implementación seria aun así poco costosa ya que constaría únicamente de un array cuyo índice seria el tipo de partición y el contenido el número de particiones de dicho tipo libres u ocupadas, la definición del tipo debería de llevar el alto y ancho de la partición.

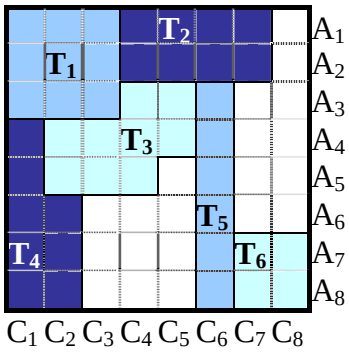


El coste del algoritmo sigue siendo lineal lo cual es una ventaja y las posibilidades geométricas también y se reduce así mismo la fragmentación.

Desventajas, siguen siendo la perdida de espacio si no esta libre ninguna partición lo suficientemente pequeña y de misma geometría.

3.2.2.2.3. Celdas de tamaño variable

Es la mejor implementación, frente a todas las demás, el problema es que el algoritmo es algo costoso y difícil de implementar, existen diversas ramas y grupos de investigación, que están desarrollando algoritmos para solucionar este problema, véase [4], nosotros básicamente solo vamos a comentarla, pero no se plantea como una gestión aprovechable en estos momentos, pero si en un futuro, ya que es la mejor apuesta.



A favor tiene, que aprovecha al máximo la FPGA, ya que se pueden definir cualquier forma geométrica basada en rectas horizontales y verticales, de esta manera elimina totalmente la fragmentación interna.

En contra, además de la difícil y costosa implementación esta la fragmentación externa, que aunque es prácticamente inevitable en casi todas las implementaciones dinámicas, se soluciona desfragmentando cuando es necesario, que aunque es un coste en tiempo es la mejor solución.

4. ORIENTACIÓN A CONDOR

4.1. ¿POR QUÉ CONDOR?

Actualmente hay varios JMS's disponibles, ya sean comerciales o de dominio público. Por ello había que ver cuál de todos ellos se adecuaba más a lo que queríamos hacer, puesto que no todos tienen las mismas características, ni siquiera todos ellos tienen una misma estructura, por ejemplo mientras que unos tienen un planificador centralizado, otros no lo tienen. Para elegir fue importante una comparativa publicada en [5], en donde se comparaban varios JMS's tales como LSF, CODINE, PBS, Condor y RES, todos ellos, sistemas con un planificador centralizado, aunque en el artículo también se menciona a los que no lo tienen. Dicha comparación consistía realmente en varias comparaciones, cada una de las cuales se fijaba en un determinado aspecto (eficiencia, seguridad, etc), pero la que más nos interesaba a nosotros era la comparativa relacionada con la posibilidad de extender el sistema elegido para que pudiese gestionar hardware reconfigurable, y en tal comparativa Condor sale bastante bien parado, a pesar de que en otros sistemas como LSF y CODINE no sería necesario, según el artículo, hacer cambios en sus códigos fuente, mientras que en Condor, a priori, habría que hacer algunos pequeños cambios para lograr las mismas funcionalidades que en esos sistemas. Por otro lado, Condor es uno de los sistemas de gestión de tareas más antiguos (de hecho su kernel no ha cambiado desde 1988), y por ello se le conoce bastante bien, ha sido bastante probado y depurado, etc. Por último, Condor es de dominio público, mientras que otros sistemas (como LSF, que en la comparación general se mostraba bastante superior al resto) son comerciales. Por tanto, la elección de Condor era bastante clara y acertada.

4.2. ¿QUÉ NOS OFRECE CONDOR?

Condor es un planificador de tareas software distribuidas, y la parte de planificador es precisamente el punto básico que nos interesa de este sistema, ya que lo que queremos es aprovechar la(s) FPGA('s) disponible(s) en la red y por tanto estaremos enviando tareas desde distintas máquinas, lo que tendrá que ser planificado convenientemente. Con lo cual las características principales de Condor que utilizaremos serán las típicas de este tipo de sistemas.

Así pues, usaremos todos sus recursos disponibles relacionados con la planificación de tareas software: su posibilidad de enviar trabajos desde cualquier máquina, la posibilidad de asignar prioridades a los trabajos que enviemos, podremos establecer prioridades a los usuarios (por ejemplo esto sería útil si queremos que un departamento tenga prioridad sobre otro), se podrá ver qué máquina es la más adecuada para la ejecución de nuestros trabajos, especificar que la máquina en la que se vaya a ejecutar cumpla determinados requisitos, etc.

Está claro que esto no es todo lo que queremos, ya que todos los sistemas de estas características pueden planificar y enviar. La otra característica principal que nos interesa de Condor es su sistema para definir y administrar los recursos, el mecanismo de ClassAds, cuyo funcionamiento explicaremos con mayor precisión en el [Apartado 5.2]. Este mecanismo simplifica enormemente tanto la tarea de añadir nuevos atributos que definan nuestros recursos, como posteriormente todo lo relacionado con la administración del recurso, la elección de cuál es el recurso al que conviene enviar el trabajo que deseemos, la imposición de determinadas restricciones y/o preferencias, etc. Y todo ello modificando realmente muy pocas cosas.

4.3. GESTIÓN DE HARDWARE IMPLEMENTADA

En realidad, dadas las características de Condor, especialmente la de los ClassAds antes ligeramente mencionada, hemos podido comprobar que es bastante sencillo implementar varios de los modelos de gestión comentados en el [Apartado 3] directamente, pero como al comenzar el proyecto no sabíamos de antemano hasta dónde podíamos llegar ni sabíamos muchas cosas acerca de este sistema, optamos por implementar el modelo más simple, esto es, la FPGA está ocupada o está libre (Gestión Estática), sin tener en cuenta particiones, bloques, ni nada parecido, y como es ese el modelo con el que hemos trabajado durante prácticamente todo el proyecto, cuando a partir de ahora hablemos de las modificaciones necesarias para ampliar Condor para la gestión de FPGA's nos estaremos refiriendo concretamente a ese modelo. En cualquier caso, como decimos, es

bastante sencillo implementar alguno de los otros modelos, y se explicarán las medidas necesarias para ello en el [Apartado 6.3].

5. CONDOR, O THE CONDOR HIGH THROUGHPUT COMPUTING SYSTEM

Puesto que el proyecto está basado en Condor, creemos que es necesario un apartado destinado a introducir algunos conceptos sobre Condor, su estructura, funciones, en definitiva, varios aspectos del sistema con los cuales se ha trabajado en el proyecto y que por tanto es necesario conocer para entenderlo. Así pues, si no conoce Condor, pensamos que los cuatro primeros puntos de este apartado le resultarán probablemente muy útiles como introducción a los aspectos básicos de este sistema. Si por el contrario ya conoce Condor, quizás prefiera pasar directamente al [Apartado 5.4], donde se explica la relación de todos estos conceptos con nuestro proyecto, o lo que es lo mismo, el uso que haremos de ellos para alcanzar nuestros fines.

5.1. ¿QUÉ ES EXACTAMENTE CONDOR?

Condor es el producto del Condor Research Project llevado a cabo en la Universidad de Wisconsin-Madison, y fue instalado por primera vez en el Departamento de Ciencias de la Computación de dicha universidad hace unos 15 años.

En propias palabras de sus creadores, Condor es un sistema de gestión de trabajo para tareas que requieren grandes cantidades de cálculo. Al igual que otros sistemas de procesamiento por lotes, Condor tiene un mecanismo de colas, política de planificación, prioridades, y monitorización y gestión de recursos. El funcionamiento buscado es el siguiente: los usuarios envían sus trabajos a Condor, y éste los mete en una cola, elige según una determinada política (así como según las especificaciones y requisitos del trabajo en cuestión) dónde y cuándo ejecutarlos, monitoriza sus progresos, y finalmente informa a los usuarios que enviaron los trabajos cuando éstos han sido completados.

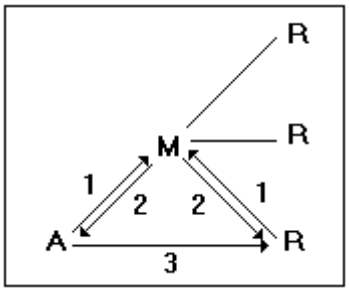
Por tanto, ¿para qué sirve Condor? Bueno, está claro que su principal objetivo son las tareas pesadas, tareas que requieren grandes cantidades de cálculo y que por tanto necesitan hacer un gran uso de las CPU's. Para sortear esta restricción de depender de un potente ordenador, un recurso muy caro, Condor puede hacer que dichas tareas se ejecuten en distintas máquinas, quizá menos potentes, pero que si sumamos sus potencias de cálculo nos pueden dar una potencia incluso mayor que un potente supercomputador. Con Condor podemos incluso aprovechar los momentos en los que algún ordenador de sobremesa no se está usando, al estilo de algunos proyectos más conocidos como el hace unos años popular SETI@home (un proyecto para analizar señales de radio recibidas desde el espacio con el fin de localizar vida extraterrestre, usando para ello Internet y ordenadores domésticos de miles de ciudadanos). Esta idea es muy atractiva, ya que además Condor nos permite establecer el significado del término "no se está usando", pues podemos definir un cierto tiempo tras el cual si, por ejemplo, el teclado está inactivo, nuestro ordenador pase a formar parte del pool de ordenadores disponibles para los cálculos antes comentados. Si volviésemos a necesitar el ordenador, no hay problema: Condor posee un sistema de checkpoints (o puntos de control) que permite dejar de ejecutar una tarea en un ordenador en un determinado punto, y retomarla en otro ordenador desde ese mismo punto. La ventaja de Condor respecto a sistemas como el del antes mencionado proyecto SETI@home es que Condor no está diseñado para un único fin (por ejemplo el sistema de ese proyecto sólo servía para analizar esas señales) y por tanto puede ser muy útil para pequeñas empresas o departamentos que necesitan realizar multitud de cálculos pero que no pueden permitirse el lujo de diseñar un sistema exclusivo para ello.

Una vez sabiendo exactamente qué es Condor y cuál es su principal utilidad, puede parecer que no es exactamente lo que se busca en este proyecto, ya que lo que nosotros buscamos inicialmente es ejecutar tareas aprovechando un recurso, o recursos, disponibles en un pequeño número en una determinada red, no grandes trabajos que requieren ingentes cantidades de cálculo. Pero hemos de fijarnos en que, por otro lado, el fin de Condor es ese, ejecutar trabajos, aprovechando también al máximo los recursos disponibles por medio de sus funciones de planificación, y esto sí nos interesa de veras. Como hemos dicho, Condor se encarga de comprobar los requisitos de cada trabajo y enviarlo a la máquina más conveniente, por tanto podremos enviar trabajos que requieran una FPGA a Condor desde cualquier máquina y el se encargará de hacer que vaya a la máquina adecuada y utilice el recurso necesario. Incluso, aparte de ir a una máquina que posea una FPGA, podremos especificar, si queremos, datos sobre el trabajo, como por ejemplo el nivel de ocupación necesario en la FPGA, y asegurar que es posible ejecutar un determinado trabajo haciendo uso de un recurso concreto. Como se puede

ver, esto está realmente de acuerdo con nuestros fines. Así que en cierto modo es verdad que no se está utilizando todo lo que Condor puede proporcionar, pero sí una parte importante y muy útil de él.

5.2. LA ESTRUCTURA DE CONDOR

Antes de entrar de lleno con la estructura de Condor, y aunque tampoco pretendemos profundizar en exceso (para una explicación detallada acerca de todo lo relacionado con Condor está el manual), expondremos un pequeño ejemplo muy básico de cómo funciona, puesto que creemos que resultará mucho más fácil de entender una vez vista en funcionamiento. Así pues, en la figura podemos ver un pool de Condor. Cada letra representa a una máquina, que en este caso están nombradas de acuerdo con el papel que desempeñan en el ejemplo: A es la máquina en la que está el agente que funcionará en el ejemplo, R son máquinas con el recurso necesario para ejecutar nuestro trabajo, y M es el cental manager, o gestor central, que es quien lleva a cabo el papel de matchmaker (o "emparejador", por así decirlo). Visto esto, el funcionamiento es el siguiente: el usuario, que está en la máquina A, desea ejecutar un trabajo que requiere un determinado recurso (disponible en las máquinas R), así que se lo envía al agente de Condor, que se encargará de almacenar el trabajo mientras encuentra alguna máquina con el recurso necesario. Esta tarea la realiza "anunciándose" al matchmaker (algo como "busco máquina disponible para ejecutar trabajo que requiere recurso X"), cosa que, por otro lado, también están haciendo las máquinas R ("máquina con recurso X se ofrece para ejecutar trabajo"), indicado en la figura con un 1. El matchmaker, entonces, basándose en todos los anuncios que le llegan, empareja los anuncios compatibles y avisa de ello a las máquinas implicadas en esa unión (2), tras lo cual el agente, que es quien almacenaba el trabajo, se pone en contacto con la máquina R del emparejamiento (3) y puede, así, ejecutarse finalmente el trabajo.



Como hemos dicho, el anterior es un ejemplo sencillo; por ejemplo, una máquina no tiene por qué tener o recursos o un agente, puede tener ambas cosas, incluso una máquina puede tener no uno sino varios agentes. Igualmente, el matchmaker no se limita a ver si los requisitos de una máquina coinciden con los de otra y ya está, puesto que además comprueba otras cosas como prioridades, si se permite el servicio a una determinada dirección IP, etc. En resumen, el ejemplo está simplificado para ver únicamente el funcionamiento de forma simple y no ha de tomarse como algo definitivo.

5.2.1. EL POOL

Una vez visto ese ejemplo, podemos pasar a ver la estructura de Condor desgranándolo un poco. Como hemos visto, lo que tenemos, principalmente, es un pool. El pool lo componen una máquina que hace el papel de central manager y un número cualquiera de otras máquinas, así pues, el pool será un conjunto de recursos (máquinas) y peticiones (trabajos). Acabamos de decir que una máquina "hace el papel de central manager". Esto nos lleva a plantearnos la pregunta "¿puede una máquina tener otros papeles?". La respuesta a esta pregunta es afirmativa, y de hecho una máquina puede tener uno de los siguientes tres papeles: central manager, ejecución, y envío. Opcionalmente existe un cuarto papel, que sería el de servidor de checkpoints. Al igual que ocurría antes con los recursos y los agentes, una máquina puede tener más de un papel, y la mayoría de ellas tienen más de uno. A continuación, pasamos a detallar brevemente cada papel:

- ***Central manager***. Como ya vimos en el ejemplo, el central manager se encargará de recolectar toda la información de las otras máquinas, y se encargará de emparejar recursos con peticiones, o dicho de otro modo, hará de negociador. Hemos dicho que "(...) el pool lo componen una máquina que hace el papel

de central manager (...)" , y por tanto sólo puede haber un central manager en el pool. Esto significa que la máquina que lleve a cabo este papel ha de ser una máquina bastante fiable, puesto que si se estropea se acabó el central manager, y ya no habrá nadie en el pool que pueda llevar a cabo las tareas de recolección de información y emparejamiento. Por otro lado hay que tener en cuenta que el central manager recibe información de todas las máquinas, ya sean informes de recursos o peticiones, así que sería deseable una buena conexión a la red.

- ***Ejecución***. Si una máquina tiene este papel quiere decir que puede ejecutar trabajos de Condor, siempre que sus recursos se adecuen a los requisitos de los trabajos, claro está. En cualquier caso, como este papel no requiere tener demasiados recursos (a menos que estemos seguros de que vamos a enviar únicamente trabajos muy exigentes) casi cualquier máquina puede ser una máquina con el papel de ejecución. Además, cuantas más máquinas con este papel haya disponibles, habrá más posibilidades para ejecutar un trabajo, incluso si la máquina no tiene demasiados recursos: puede que haya trabajos que no requieran demasiado y encuentren en esta máquina un buen complemento.

- ***Envío***. Si una máquina tiene este papel quiere decir que puede enviar trabajos de Condor. Si decíamos en el punto anterior que prácticamente cualquier máquina podía tener el papel de ejecución, con el papel de envío no ocurre exactamente lo mismo, pues si bien cualquier máquina puede tener este papel, en este caso sí que hacen falta ciertos recursos, básicamente de disco, puesto que al enviar un trabajo se genera un proceso en la propia máquina desde la que se envía (ya vimos el agente en el ejemplo), aparte de que el mecanismo de checkpoints del que se habló en el [Apartado 5.1] hace que se almacene información en la máquina que envía, aunque este segundo problema se puede aliviar un poco con el papel opcional del que hablamos anteriormente: el servidor de checkpoints.

- ***Servidor de checkpoints***. La máquina con este papel, opcional como hemos dicho, es una máquina con la función de almacenar todos los archivos de checkpoint de los trabajos que se han enviado en el pool. Al igual que el central manager, esta tarea la desempeña una máquina y ha de tener igualmente una buena conexión a la red, aparte de una importante cantidad de espacio en disco para almacenar todos esos archivos.

Para desempeñar estos papeles, Condor utiliza varios demonios, que se encargarán de llevar a cabo las tareas que acabamos de comentar para cada papel. Así, en cada máquina del pool tendremos uno o varios de estos demonios:

- ***condor_master***. Este demonio se encarga de hacer que el resto de demonios necesarios para Condor se estén ejecutando, y de vez en cuando comprueba si alguno de ellos tiene alguna novedad, en cuyo caso lo reinicia. Por tanto, este demonio se ejecuta en todas y cada una de las máquinas del pool, tengan el papel que tengan.

- ***condor_startd***. Este demonio representa a una determinada máquina, con ciertos atributos con información acerca de los recursos de que dispone, en el pool. Así que este demonio se tendrá que ejecutar en cualquier máquina que desempeñe el papel de ejecución, ya que es el encargado, por así decirlo, de anunciar que hay una máquina X con recursos Y en el pool, por si alguien la necesita. Este demonio se ayuda de otro cuando está listo para ejecutar un trabajo, el demonio *condor_starter*, que se encarga de ejecutar realmente el trabajo (además de proporcionarle lo que necesite y monitorizarlo) y de enviar de vuelta los resultados a la máquina que lo envió.

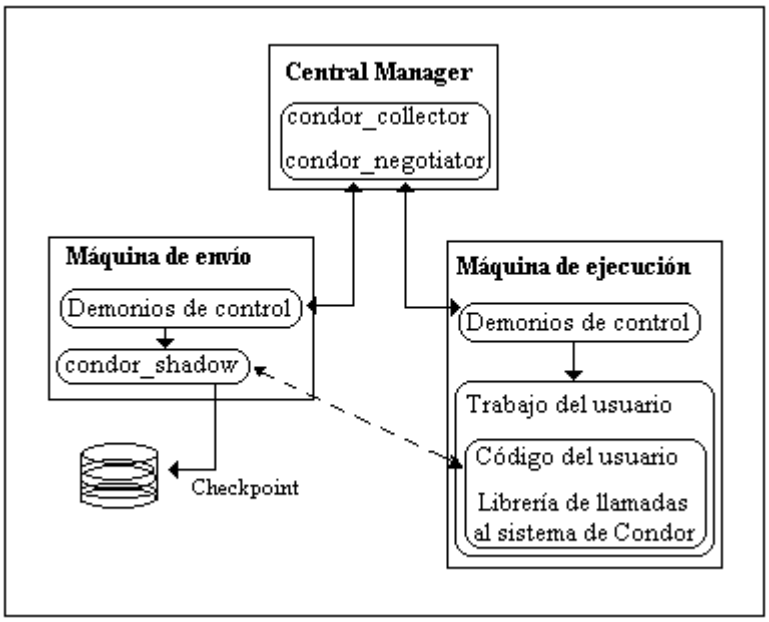
- ***condor_ schedd***. Si el demonio condor_startd representa los recursos, *condor_schedd* representa las peticiones. Por tanto este demonio tendrá que estar ejecutándose en cualquier máquina que desempeñe el papel de envío. Su función es almacenar los trabajos que se envían en una cola de trabajos, e ir anunciando dichos trabajos por si hay alguna máquina que pueda ejecutarlos. Al igual que *condor_startd* se apoya en *condor_starter*, *condor_schedd* se apoya en *condor_shadow* para completar la tarea, ya que *condor_shadow* se encarga de gestionar el trabajo remotamente una vez se ha comenzado una ejecución en una máquina remota. Esto es, cualquier cosa que necesite el trabajo de la máquina desde la que se mandó, se la proporcionará *condor_shadow*.

- ***condor_collector***. Dijimos antes que el central manager llevaba a cabo dos tareas: recolectar información y emparejar recursos con peticiones. *condor_collector* se encarga de la primera de estas dos tareas, y por tanto se ejecutará en la máquina que desempeñe el papel de central manager. Así que este demonio estará recogiendo toda la información acerca del estado del pool, ya que los demás demonios antes vistos le estarán enviando periódicamente sus anuncios acerca de recursos disponibles o solicitados.

- ***condor_negotiator***. Al igual que *condor_collector* se encarga del trabajo del central manager relacionado con la recolección de información, *condor_negotiator* se encarga de la otra función básica del central manager: el emparejamiento entre recursos y peticiones. Su funcionamiento es el siguiente: cada cierto tiempo le pregunta a *condor_collector* para ver cuál es el estado del pool, y en función de esa información, se pone en marcha para emparejar recursos con peticiones. Además, es el encargado de gestionar las prioridades de los distintos usuarios y trabajos, prioridades que utiliza al hacer los emparejamientos.

Así pues, los tres primeros demonios suelen estar ejecutándose en la mayoría de las máquinas, ya que generalmente se suelen utilizar todas las máquinas del pool tanto con el papel de ejecución como con el de envío. En cambio, como es lógico, los dos últimos demonios sólo están presentes en la máquina que hace de central manager. Al hablar de los distintos papeles que podía desempeñar una máquina, hablamos de un papel opcional, el de servidor de checkpoints. En caso de que hubiese una máquina desempeñando este papel, esta máquina ejecutaría el demonio *condor_ckpt_server*, que lleva a cabo las funciones descritas antes para este papel.

Vistos tanto los papeles que puede tomar una máquina como la forma de desempeñarlos, la arquitectura del pool de Condor sería más o menos así:



5.2.2. EL MECANISMO DE CLASSADS

Bien, ya hemos visto la estructura del pool y el comportamiento de las máquinas para enviar trabajos y ejecutarlos remotamente, y se hemos podido comprobar la importancia que tienen en todo este proceso los anuncios de las máquinas tanto para ofrecer sus recursos como para pedir la ejecución de un trabajo utilizando recursos de otras máquinas, pero ¿en qué consisten estos anuncios exactamente? A decir verdad, el uso que hemos venido haciendo hasta este punto de la palabra "anuncio" para referirnos a la forma que tienen las máquinas de avisar al central manager no es fortuito, puesto que la forma en que está concebida la idea está tomada precisamente de los anuncios, y de ahí el nombre del mecanismo: ClassAds, Classified Advertisements, o lo que es lo mismo, Anuncios Clasificados.

Ya hablamos en el [Aapartado 4], muy de pasada, acerca de los ClassAds. Dijimos que nos simplificarían la tarea de definir y gestionar nuevos recursos, y esto es así porque este mecanismo esta diseñado de hecho para simplificar todo lo relacionado con el emparejamiento de recursos y peticiones. Acabamos de decir que tanto la idea como el nombre están tomados de la sección de anuncios clasificados del periódico, y es exactamente así como funciona el mecanismo. En un periódico, los vendedores y los compradores se anuncian en él. Los vendedores ponen lo que ofrecen, con sus detalles, etc., por ejemplo si venden una casa pondrán cuántas habitaciones tiene, los metros cuadrados, el número de baños, si tiene calefacción, ascensor, garaje, etc. Y, como no, un precio. Por otro lado, el comprador expone lo que desea comprar, así, siguiendo con el ejemplo de la casa, alguien podría poner un anuncio buscando una casa mayor de X metros cuadrados, con más de dos habitaciones, etc., y un precio máximo que estaría dispuesto a pagar. Por último, tanto vendedor como comprador tirarán siempre hacia lo más provechoso, esto es, el comprador irá a por la casa más barata que satisfaga sus deseos y el vendedor se interesará primero en la persona que más dinero le ofrezca.

Pues bien, si llevamos esto al campo de los ordenadores, está todo hecho. Cambiemos las casas por ordenadores, y pongamos que alguien desea vender su ordenador mientras que otra persona está interesada en comprar un ordenador que tenga cierta cantidad de RAM, una determinada CPU, tanto espacio en disco... De igual modo se anuncian las máquinas en Condor: cada máquina anuncia en el pool la cantidad de RAM disponible, el espacio con el que cuenta libre en el disco duro, el sistema operativo, y muchos datos más tales como el tipo de trabajos que prefiere, condiciones que deben cumplir los trabajos que le lleguen, etc. Si nos ponemos en el otro lado, al enviar un trabajo podemos indicar el tipo de máquina que necesitamos, con una determinada cantidad de memoria, disco, o lo que queramos, y también podemos incluir preferencias sobre lo que más nos conviene. Condor está continuamente leyendo todos los ClassAds, y así puede ir emparejando recursos con peticiones, incluso teniendo en cuenta qué es lo más provechoso en cada caso.

5.2.3. LOS UNIVERSOS

Hemos visto el pool con los papeles que desempeña cada máquina y cómo éstas anuncian sus recursos y envían sus trabajos. Así que nos queda precisamente ver cómo se envían y se ejecutan los trabajos en Condor. Pero para ello, antes de pasar a ver los trabajos en sí, tenemos que hablar un poco de los universos existentes en Condor. Los universos son los entornos de ejecución de Condor, entre los cuales hay que elegir uno para que se ejecute en él el trabajo que queremos. Cada universo tiene algunas particularidades y están diseñados con distintos fines, por tanto es importante elegir bien el universo bajo el que se ejecutarán nuestros trabajos.

Hay varios tipos de universos en Condor: standard, vanilla, java, globus, PVM, MPI, y scheduler. En este punto nos limitaremos a comentar los tres primeros: los dos primeros porque son los básicos, y el universo java porque nos resultará útil en nuestro proyecto. Antes de ver estos tres tipos de universo, simplemente comentar que cuando elijamos un universo tenemos que especificarlo a la hora de enviar el trabajo (cosa que trataremos en el siguiente punto). Si no lo hacemos, Condor entenderá que se desea utilizar el universo standard, que es el universo por defecto.

- ***Standard***. Como hemos dicho, es el universo por defecto. Esto es porque puede aprovechar dos interesantes funcionalidades de Condor, y así aprovechar todo lo que este sistema ofrece: checkpoints y llamadas al sistema remotas. Ya explicamos lo que son los checkpoints, así que explicaremos las llamadas, aunque ya hablamos de ellas al hablar del demonio *condor_shadow*. En ese momento dijimos que provee al trabajo que se ejecuta remotamente todo lo que necesita, así, todas las llamadas al sistema que realice el trabajo, las atenderá *condor_shadow* y le proporcionará la respuesta; por ejemplo, si el trabajo remoto necesita un archivo que está en la máquina desde la que se envió y llama al sistema preguntando por él, es *condor_shadow* quien atenderá la petición, lo buscará, y se lo proporcionará. Si se desea utilizar este universo, primero hay que utilizar el comando *condor_compile* para enlazarlo con las librerías de Condor. Hay que hacer notar que no todos los trabajos pueden ser re-enlazados con las librerías de Condor. En ese caso, lo mejor es utilizar el universo vanilla.

- ***Vanilla***. Este universo está pensado para aquellos trabajos que no pueden ser re-enlazados con las librerías de Condor. Se trata más bien de una alternativa puesto que a diferencia del universo standard no puede hacer uso ni de los checkpoints ni de las llamadas al sistema remotas, con los inconvenientes que ello conlleva, esto es, los trabajos que no se han completado tendrán que esperar a que el recurso

esté libre nuevamente, o comenzar en otra máquina desde cero, y habrá que suministrar todo lo que el trabajo pueda necesitar en la máquina remota.

- ***Java***. En caso de que usemos este universo, Condor se encargará de todos los detalles relacionados con la máquina virtual de Java (JVM), como por ejemplo establecer el classpath. Así que resulta idóneo para ejecutar tareas escritas para la JVM.

5.2.4. LOS TRABAJOS

Ahora que ya sabemos qué son los universos y su utilidad, pasamos al último apartado de este punto en el que trataremos el elemento que nos queda por detallar de lo que vimos en el ejemplo: los trabajos. Hemos dicho anteriormente que tanto las máquinas como los trabajos se anunciaban con los ClassAds; en las máquinas ya hemos visto cómo se hacía puesto que simplemente se limitan a poner esa información a disposición del central manager durante todo el tiempo, pero ¿cómo se anuncian los trabajos, que están continuamente llegando y saliendo de la cola de trabajos? Está claro que si se envía un trabajo a Condor tal cual, no sabrá realmente qué necesita ese trabajo para ejecutarse, es decir, qué cantidad de recursos necesita. Por tanto al enviar un trabajo tendremos que proporcionar a Condor esa información. Esto se hace por medio de un archivo llamado submit description file, o archivo de descripción de envío, que contendrá toda la información necesaria para que Condor sepa cómo gestionar el trabajo (requisitos, preferencias, ubicación del ejecutable...), aparte de información acerca de dónde puede encontrar los datos necesarios para la ejecución del mismo, si se desea monitorizar el trabajo llevando un log, etc. En resumen, en este archivo tiene que estar todo lo que se puede necesitar saber acerca del trabajo, ya que es este archivo lo que en realidad enviamos a Condor para ejecutarlo, mediante el comando condor_submit. Al ejecutar el comando anteriormente mencionado con el descriptor como argumento, se crea un ClassAd para el trabajo, y será con él con el que trabaje Condor.

La estructura de este tipo de archivos es muy sencilla, ya que consisten simplemente en la información antes indicada, así que es bastante intuitivo puesto que es como dar unas pequeñas instrucciones sobre cómo poner un anuncio en el periódico: lo que quiero ejecutar, los requisitos, algún detalle sobre cómo quiero que se lleve a cabo el trabajo... A continuación ponemos un par de ejemplos para mostrar cómo son estos archivos; para más ejemplos e información precisa acerca de los parámetros que se usan en los archivos puede consultar el manual de Condor.

En el primer ejemplo supongamos que queremos ejecutar un programa “cálculos” que realiza diversos cálculos con unos datos que le suministramos por la que sería la entrada estándar y nos devuelve esos datos por pantalla. Además, nuestro programa necesita ejecutarse en máquinas con más de 32 megabytes de memoria y sólo funciona en Linux. Por último, también queremos que se lleve un log acerca de la ejecución del trabajo. El descriptor entonces quedaría así:

```
########################
#  Ejemplo de cálculos #
########################

Executable = calculos
Requirements = Memory >= 32 && OpSys == “LINUX”
Rank = Memory >= 64
Input = calculos.data
Output = calculos.out
Error = calculos.error
Log = calculos.log
Queue
```

Primero le hemos indicado cuál es el archivo que debe ejecutar, “calculos”. A continuación, como queríamos que la máquina en la que se va a ejecutar cumpliese unos determinados requisitos, se lo indicamos con la variable Requirements. También hemos especificado después, con Rank, que si es posible ejecutar el programa en una máquina cuya memoria sea mayor o igual que 64, lo ejecute preferiblemente en dicha máquina, esto es, Rank simplemente es para indicar preferencias (como ya dijimos al hablar de que tanto vendedores como compradores suelen tener preferencias). Y después, suministramos la información referente a la entrada y salida,

puesto que hemos dicho que nuestro programa lee y escribe, respectivamente, de la entrada estándar y en la salida estándar, pero Condor no permite interactividad y además, si se ejecuta en otra máquina, no va a escribir en la pantalla de esa máquina unos resultados que sólo nos interesan a nosotros. Así que lo que hacemos con las variables Input, Output, y Error, es redirigir stdin, stdout, y y stderr a diferentes archivos. Como queríamos un log, también le hemos indicado un archivo en el que escribir el log. Por último, mandamos el programa a la cola de trabajos.

Es importante darse cuenta de que en ninguna parte le hemos dicho bajo qué universo tiene que ejecutarse nuestro programa, así que Condor entenderá que hay que ejecutarlo en el universo standard, y por tanto tendremos que haber utilizado el comando condor_compile para que utilice las librerías de Condor, algo necesario en este universo. Si queremos usar otro universo tendremos que especificarlo explícitamente, como en este segundo ejemplo:

```
####################
# Ejemplo de Java #
####################

Universe = java
Executable = hola.class
Arguments = hola
Output = hola.output
Error = hola.error
Log = hola.log
Queue
```

En este ejemplo queremos ejecutar el típico programa de “Hola Mundo”, nos da igual en qué máquina así que no especificamos requisitos, eso sí, tendrá que ser alguna máquina que tenga la JVM. Como es un trabajo diseñado específicamente para Java, seleccionamos el universo java. Ahora el ejecutable es un .class, y hay que indicarle a la JVM el qué es lo que queremos ejecutar, así que en Arguments le damos el comando que ejecutaríamos; si tuviese argumentos (por ejemplo, dos números) pues habría que escribir “hola 3 4”. El resto de variables son las vistas anteriormente. Hemos escogido este ejemplo porque así podemos mostrar el uso con otros universos y en concreto para el universo java, que es con lo que hemos trabajado en este proyecto.

Una vez hemos ejecutado el comando condor_submit con el archivo de descripción del envío, el trabajo pasa a la cola, de donde los cogerá Condor para intentar emparejarlo con los recursos necesarios. A partir de este momento, podemos ver el estado del trabajo, o también tenemos diferentes opciones tales como eliminar el trabajo de la cola, pausarlo, o incluso cambiar la prioridad del trabajo (que no prioridad del usuario; la prioridad del trabajo indica a Condor cuál de todos los trabajos de un usuario es más prioritario, pero la prioridad del usuario es la misma).

Cuando al fin se ejecuta el trabajo, Condor lo eliminará de la cola, lo insertará en su historial, y avisará al usuario que lo mandó ejecutar de la forma que éste haya especificado. Si le indicamos que llevase algún log o que nos devolviese resultados (como en los ejemplos que vimos antes) también obtendremos dichos archivos.

5.3. FUNCIONES Y COMANDOS IMPORTANTES

En este apartado comentaremos algunas de las funciones de Condor que utilizaremos, y los comandos necesarios para ello. Condor dispone de muchas funciones y tanto ellas como sus respectivos comandos pueden ser consultados en el manual de Condor. En este apartado no pretendemos hacer una descripción detallada de cómo utilizar Condor, sino describir brevemente aquellos comandos que consideramos útiles para el uso de Condor para enviar tareas que han de ejecutarse en una FPGA disponible en la red, en definitiva, una especie de guía rápida básica para ejecutar trabajos.

Lo primero, obviamente, es arrancar Condor. Como dijimos anteriormente, si y sólo si hay un único central manager Condor funciona. Por ello, aparte de arrancar Condor en la máquina que usemos, hay que arrancarlo antes en el central manager. Tanto en una máquina como en otra esta operación se lleva a cabo con el comando **condor_master**, y su función es la de lanzar al demonio “condor_master”comentado en el [Apartado

5.2.1], que es el que se encarga de mantener despiertos a todos los demonios necesarios para la ejecución de Condor.

Igualmente que lo primero es encender Condor, lo último será apagar Condor, para lo que usaremos el comando **condor_off –master**, que apaga todos los demonios, condor_master incluído. Es importante el argumento –master, puesto que si no se pone, apaga todos los demonios pero no al demonio condor_master, y por tanto podrá seguir manejando a otros demonios, pudiendo incluso despertarlos si fuese necesario, tarea que se realiza con el comando **condor_on**.

Una vez hemos arrancado Condor en las distintas máquinas, tras unos segundos que tardan en encontrarse unas a otras con sus respectivos ClassAds, estados, etc., Condor está preparado para que se le envíen tareas. Pero dejemos el envío de tareas para un poco más adelante. Por ejemplo, es posible que antes de enviar queramos ver qué máquinas hay en el pool en ese momento. Esto lo podemos hacer con el comando **condor_status**. Al ejecutar este comando se nos mostrarán todas las máquinas del pool junto con información sobre, entre otras cosas, su estado, o su arquitectura. Además de esta información, Condor nos ofrece la posibilidad de conocer la información sobre los ClassAds de cada una de estas máquinas, para lo que tendremos que añadir el parámetro **–l** (esto es, sería ejecutar el comando **condor_status –l**). Esto es muy útil para saber cuál es el ClassAd, con sus atributos, de una máquina determinada. Sin embargo en cuanto el pool es un poco grande puede ser un caos encontrar en la lista la máquina que nos interesaba. Por un lado, puede que lo que estemos buscando es saber si hay una máquina con un determinado atributo en su ClassAd. En ese caso tendremos que utilizar el parámentro –constraint. Así, por ejemplo, si lo que buscamos es una máquina con Linux, tendremos que ejecutar el comando: **condor_status –constraint OpSys==\"LINUX\"**. Es necesario fijarse en las barras delante de las comillas, ya que normalmente son necesarias. Por otro lado, quizás lo que buscábamos era el estado de una máquina en particular, o incluso el estado de un determinado atributo. En ese caso es más útil el comando **condor_config_val**.

El comando **condor_config_val** devuelve el valor del atributo que pongamos como parámetro. Por ejemplo, si queremos saber el valor de nuestro atributo OpSys, pondremos **condor_config_val OpSys**. Con el parámetro **–set** podemos además configurar el valor a la variable que indiquemos, deshacer el cambio con **–unset**, o con el parámetro **–name** obtener el valor del atributo del ClassAd de otra máquina, entre otras varias opciones que nos ofrece este comando. Se ve claramente que este comando no debería ser utilizado por cualquier usuario, ya que permitiría entonces modificar los ClassAds libremente incluso con informaciones erróneas, así que requiere el nivel de acceso CONFIG para poder usarse. Por otro lado, ningún cambio tiene efecto hasta que no se ejecuta el comando **condor_reconfig**.

El comando **condor_reconfig** se encarga de reconfigurar a los demonios de acuerdo con los archivos de configuración de Condor, esto es, los demonios leen de nuevo la información que necesiten de esos archivos. Por tanto si dichos archivos han sido modificados, los demonios estarán contando con nueva información y Condor se comportará de forma acorde a ella. Así que este comando se suele utilizar cuando se ha hecho algún cambio en algún atributo, como por ejemplo después de usar el antes mencionado **condor_config_val –set**.

Bueno, ha llegado la hora de enviar el trabajo. Para ello, como ya se vio anteriormente, es necesario utilizar el comando **condor_submit**, seguido del nombre del archivo descriptor del trabajo que se envía, que deberá contener la información comentada en el [Apartado 5.2.4]. Una vez se ha enviado el archivo, se almacena en la cola de trabajos de Condor, como ya vimos. Así que sería interesante poder consultar la cola de trabajos. Esta tarea se realiza con el comando **condor_q**. Si no especificamos ningún parámetro, nos mostrará la cola de trabajos local. Por el contrario, si utilizamos el parámetro **–global**, obtendremos la información de todas las colas de trabajos de todos los usuarios. Y si lo que queremos es obtener la cola de un determinado schedd, lo especificaremos con el parámetro **–name**. Y también podremos obtener una lista de todos los trabajos enviados por un determinado usuario usando el parámetro **–submitter**. Un parámetro de este comando que hemos encontrado también bastante útil es el parámetro **–analyze**. Al ejecutar el comando con este parámetro, se lleva a cabo un análisis aproximado para determinar cuántos recursos están disponibles para ejecutar trabajos, si éstos han sido rechazados, si hay máquinas que dan prioridad a otros trabajos, si hay algún trabajo ejecutándose...

Al igual que podemos enviar trabajos a la cola, podemos manipularlos allí. Así, podemos poner un trabajo “en pausa” con el comando **condor_hold**, y “despausarlo” después con el comando **condor_release**. Esto es bastante útil, por ejemplo, en el caso de que tengamos trabajos más prioritarios que otros, así, como alternativa a modificar la prioridad del trabajo, podemos pausar aquellos trabajos de menor prioridad, de modo

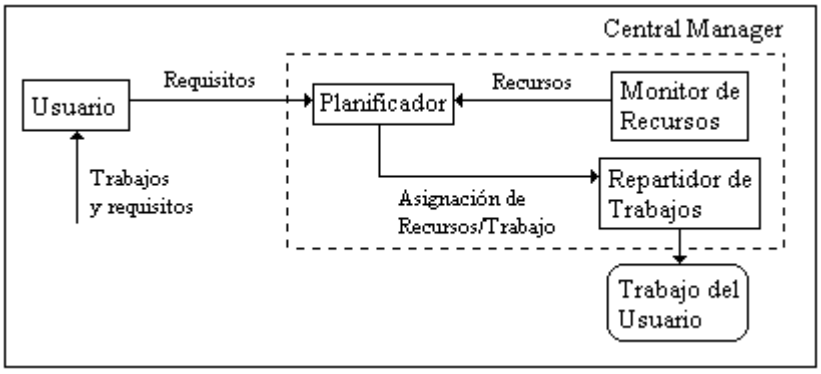
que cuando se nos conceda el turno para ejecutar, se ejecute el programa más prioritario, que no ha sido pausado. Una vez se ha ejecutado, podemos liberar ("despausar") los demás. Si realmente preferimos sólo modificar la prioridad del trabajo, entonces usaremos **condor_prio** y asignarmos la prioridad que queramos. Esta prioridad está en el rango que va desde –20 a +20 (el – y el + son necesarios), siendo mayor la prioridad cuanto mayor es el número. La prioridad por defecto es 0.

Por último, podemos también eliminar los trabajos de la cola. Para ello usaremos el comando **condor_rm**, seguido del número de clúster del trabajo si queremos eliminar todos los trabajos de ese clúster, el número de clúster seguido del nombre del proceso si sólo queremos eliminar un determinado trabajo, o incluso el nombre del usuario si lo que queremos es eliminar todos los trabajos de un determinado usuario. Si lo que queremos es eliminar todos los trabajos de la cola, habrá que utilizar el parámetro **–all**.

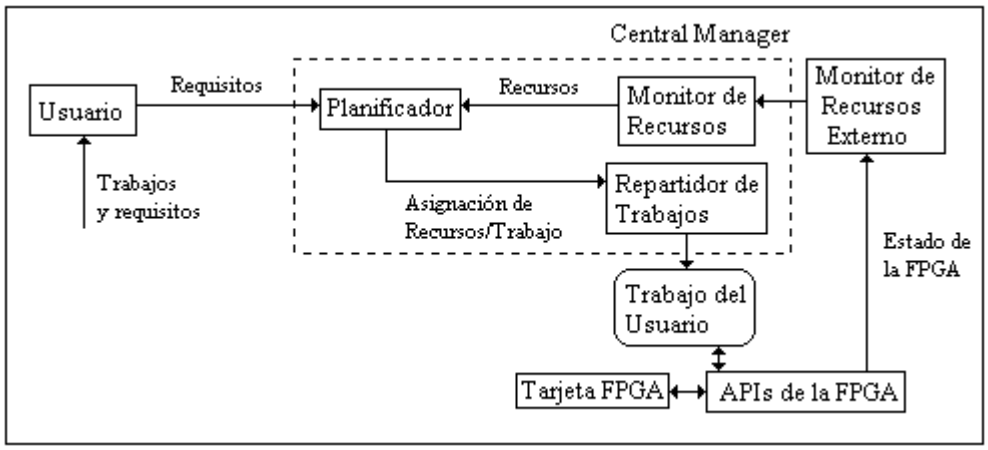
Como hemos dicho al principio, hay más comandos, e incluso cada uno de los comandos aquí nombrados tienen más opciones de las aquí comentadas, pero hemos puesto las que hemos encontrado más útiles a la hora de trabajar con Condor en nuestro caso específico. Para más información, nos volvemos a remitir al manual de Condor.

5.4. USO QUE HAREMOS DE CONDOR.

En este último apartado explicaremos qué es lo que haremos exactamente una vez que ya conocemos las distintas funcionalidades y carácterísticas de Condor, y cómo usaremos concretamente dichas funcionalidades y características para lograr nuestro objetivo. Básicamente lo que tenemos es algo así:



Y queremos obtener un sistema así:



Esto es, nuestra meta es poder gestionar también FPGA's de modo que sea posible para distintos usuarios en una red utilizar dichos recursos hardware. Pero hemos visto que Condor lo que hace es (aparte de planificar, decidir qué máquina es la más adecuada para un trabajo, etc.) ejecutar trabajos, o sea, tareas que se puedan ejecutar, en otras palabras, lo que sería un .exe. Así que nuestra idea de enviar los trabajos para las

FPGA's, consistentes básicamente en ciertos datos, no es posible de realizar, puesto que eso no es ningún ejecutable. Pero sí que es posible enviar un programa que cuando se ejecute, cosa que Condor sí puede hacer, se encargue de suministrar esos datos a la FPGA (o mejor dicho, al programa que gestiona localmente la FPGA, "APIs de la FPGA" en la figura) y se realice así la tarea. El inconveniente es que lo que realmente es un trabajo de Condor es dicho programa, y una vez se ejecute y suministre los datos a la FPGA, Condor ha cumplido su misión y por tanto no obtendremos de vuelta los datos de la ejecución en la FPGA, pero eso no entraba en los planes de nuestro proyecto.

Bien, ya tenemos decidido cómo haremos para que ejecute los trabajos, y por tanto lo que enviaremos será dicho ejecutable con los archivos (o su ruta, si utilizamos un sistema de ficheros compartido en la red), esto es, se especificará eso en un archivo descriptor del envío. Pero claro, en ese archivo también tendremos que especificar que la máquina que lo ejecute ha de tener una FPGA y, por tanto, tendremos que definirlas de algún modo en Condor. Está claro que los ClassAds son idóneos para ello, pero lo que no es tan simple es mantener informado a Condor del estado actual de la FPGA, ya que éste puede estar cambiando contínuamente según van llegando trabajos y según van ejecutándose. Por tanto tendremos que hacer algún programa que pueda comunicar estos cambios a Condor, que sería lo que en la figura de arriba es el "monitor de recursos externo".

Así pues, el trabajo consistirá en lo siguiente: una vez instalado Condor y configurado correctamente, habrá que añadir de algún modo nuevos atributos a los ClassAds para gestionar FPGA's. En nuestro caso pensamos que lo mejor sería usar dos nuevos atributos para los ClassAds: uno para indicar si la máquina tiene o no FPGA (al estilo del atributo "HasJava" que indica si la máquina dispone de máquina virtual de Java) y otro que indique el estado actual de la FPGA, ya que puesto que los atributos no tienen un tipo determinado podemos poner "TRUE" y "FALSE" si sólo usamos la FPGA como usada/libre, o un número para indicar el número de celdas libres (o usadas, según convengamos), etc., según el modelo que estemos usando. A continuación, tendremos que hacer algún pequeño programa o script que se encarque de hacer el papel de "monitor de recursos externo" que se encargue de avisar a Condor de los cambios de estado de la FPGA. Para ello, si se queremos enlazar este proyecto con el proyecto global habrá que añadir también alguna funcionalidad al gestor local de la FPGA para que se comunique con dicho programa o script. Nuestra idea es, para comprobar su funcionamiento, hacer un pequeño programa que simule ese aspecto que nos interesa del gestor de la FPGA, de modo que se pueda probar el funcionamiento del nuevo sistema sin tener que tocar nada del gestor de la FPGA. Por último, habrá que programar el ejecutable que se enviará a Condor para que, cuando éste lo ejecute, tome los datos indicados en el descriptor de envío y los haga llegar al gestor de la FPGA. Llamaremos a este programa "cargador", puesto que su función es cargar los trabajos en las FPGA's.

Por tanto, a grandes rasgos, el proyecto consistirá en: instalar y configurar Condor, cambiar sus ficheros de configuración para añadir nuestros atributos de FPGA, hacer un script que se encargue de mantener a Condor informado del estado actual de la FPGA, hacer un cargador para los trabajos que pueda ejecutar Condor, y hacer un pequeño simulador del gestor de la FPGA. Todo esto lo detallamos en el [Apartado 6].

6. CAMBIOS EN CONDOR PARA EL USO DE TAREAS HW

6.1. INSTALACIONES, CONFIGURACIONES Y AJUSTES NECESARIOS

6.1.1. INSTALACIÓN DE CONDOR EN LINUX

6.1.1.1. Preparación previa a la instalación

Lo primero que se hizo fue crear un usuario y un grupo de usuarios llamado “condor” en todas las máquinas.

También podíamos haberlo ejecutado como root directamente, pero como sabemos que esto no siempre es posible, lo hicimos con el usuario “condor”, aunque esto puede tener problemas de seguridad y rendimiento. Estos problemas se pueden solucionar con unos pequeños ajustes.

Se prepara el home del usuario “condor” con la siguiente estructura: En el directorio “~condor/Condor_Source/” copiamos el archivo fuente (condor-6.4.7-linux-x86-glibc22-dynamic.tar.gz) de Condor para realizar la instalación. A continuación descomprimimos el archivo con gunzip, y a continuación expandimos con tar. Esto generó archivos de ayuda, y la carpeta “Condor-6.4.7” donde se encuentran los archivos de instalación.

Se ejecuta “condor_install” que es un script escrito en perl. Para que se pueda ejecutar el script es necesario que perl esté instalado.

6.1.1.2. Instalación del Central Manager

A continuación se detallan los pasos que se siguieron en la instalación, asi como las respuestas dadas a las preguntas hechas:

```
Welcome to condor_install.  You are going to need to answer a few
questions about how you want Condor configured on this machine, what
pool(s) you want to join, and if this machine is going to serve as
the Central Manager for its own pool.  If you are unsure about how to
answer any of the questions asked here, please consult the INSTALL
file or the Installation chapter of the Condor Administrator's
Manual.

The installation is broken down into various steps.  Please consult
the INSTALL file to refer to a specific step if you have trouble with
it.

For most questions, defaults will be given in []'s.  To accept the
default, just press return.

If you have problems installing or using Condor, please consult the
Condor Administrator's Manual, which can be found on the World Wide
Web at: http://www.cs.wisc.edu/condor/manual/ If you still have
problems, send email to condor-admin@cs.wisc.edu.

Press enter to begin Condor installation

*****************************************************************
STEP 1: What type of Condor installation do you want?
*****************************************************************
```

```
Would you like to do a full installation of Condor? [yes] y

Press enter to continue.
```

Hay tres tipos de instalación: submit-only, full-install y central-manager. La instalación submit-only sólo permite enviar tareas para que sean ejecutadas en otras máquinas. Full-install permite enviar y ejecutar tareas. Y la instalación central-manager se realiza sobre una máquina que tenga condor instalado, para hacer que esta máquina pase a ser el central manager.

Elegimos full-install para que reciba y envíe tareas.

```
*********************************************************************
        STEP 2: How many machines are you setting up for Condor?
*********************************************************************

Are you planning to setup Condor on multiple machines? [yes] y
Will all the machines share files via a file server? [yes] n

You will need to run condor_install locally on each machine.
Setting up host "ghadir2" for Condor.

Press enter to continue.
```

Le indicamos que vamos a instalar Condor en más de una máquina, y que las máquinas no tienen un sistema de ficheros compartido entre ellas. Al principio, no sabíamos que íbamos a necesitar un sistema de ficheros compartidos, así que en este paso le indicamos que no tenían sistema de ficheros compartidos. Más tarde, cuando optamos por usar un sistema de ficheros compartidos, se instaló, y no hizo falta realizarle ningún cambio a Condor.

```
*********************************************************************
        STEP 3: Install the Condor "release directory", which holds
various binaries, libraries, scripts and files used by Condor.
*********************************************************************
which: no condor_config_val in
(/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/X1
1R6/bin:/root/bin)

I can't find a complete Condor release directory.

Have you installed a release directory already? [no] n

Where would you like to install the Condor release directory?
[/home/condor/condor] /home/condor/Condor_Release
Installing a release directory into /home/condor/Condor_Release ...
etc/
etc/examples/
etc/examples/README
etc/examples/condor.generic
etc/examples/condor_config.local.central.manager
etc/examples/condor_config.local.dedicated.resource
etc/examples/condor_config.local.dedicated.submit
etc/examples/condor_config.generic
etc/examples/condor_config.root.generic
```

```
etc/examples/condor_config.submit.generic
etc/examples/condor.boot
include/
include/user_log.README
include/user_log.c++.h
include/condor_event.h
include/file_lock.h
include/condor_constants.h
lib/
lib/libcondorapi.a
lib/libcondorc++support.a
lib/condor_rt0.o
lib/libcondorsyscall.a
lib/libz.a
lib/libcondorzsyscall.a
lib/ld
lib/real-ld
lib/libc.a
lib/libnss_files.a
lib/libnss_dns.a
lib/libresolv.a
lib/Condor.pm
lib/CondorJavaWrapper.class
lib/CondorJavaInfo.class
lib/scimark2lib.jar
bin/
bin/condor_config_val
bin/condor_userprio
bin/condor_findhost
bin/condor_history
bin/condor_qedit
bin/condor_version
bin/condor
bin/condor_checkpoint
bin/condor_vacate
bin/condor_reschedule
bin/condor_stats
bin/condor_status
bin/condor_dagman
bin/condor_submit_dag
bin/condor_userlog
bin/condor_q
bin/condor_rm
bin/condor_hold
bin/condor_release
bin/condor_submit
bin/condor_prio
bin/condor_compile
bin/condor_run
bin/condor_glidein
sbin/
sbin/condor_preen
sbin/condor_advertise
sbin/condor_fetchlog
sbin/condor
sbin/condor_on
sbin/condor_off
sbin/condor_checkpoint
sbin/condor_vacate
```

```
sbin/condor_restart
sbin/condor_reconfig
sbin/condor_reconfig_schedd
sbin/condor_reschedule
sbin/condor_master_off
sbin/condor_shadow.std
sbin/condor_starter.std
sbin/condor_startd
sbin/condor_schedd
sbin/condor_negotiator
sbin/condor_collector
sbin/condor_master
sbin/condor_install
sbin/condor_init
sbin/condor_gridmanager
sbin/gahp_server
sbin/condor_starter
sbin/rsh
sbin/condor_shadow
man/
man/man1/
man/man1/condor_checkpoint.1
man/man1/condor_compile.1
man/man1/condor_config_val.1
man/man1/condor_findhost.1
man/man1/condor_glidein.1
man/man1/condor_history.1
man/man1/condor_hold.1
man/man1/condor_master.1
man/man1/condor_master_off.1
man/man1/condor_off.1
man/man1/condor_on.1
man/man1/condor_preen.1
man/man1/condor_prio.1
man/man1/condor_q.1
man/man1/condor_qedit.1
man/man1/condor_reconfig.1
man/man1/condor_reconfig_schedd.1
man/man1/condor_release.1
man/man1/condor_reschedule.1
man/man1/condor_restart.1
man/man1/condor_rm.1
man/man1/condor_run.1
man/man1/condor_stats.1
man/man1/condor_status.1
man/man1/condor_submit.1
man/man1/condor_submit_dag.1
man/man1/condor_userlog.1
man/man1/condor_userprio.1
man/man1/condor_vacate.1
done.

Using /home/condor/Condor_Release as the Condor release directory.

Press enter to continue.
```

Nos pregunta si está instalado el directorio Condor-Release, y le indicamos que no porque es la primera vez que se instala Condor. A continuación se instala el directorio.El directorio Condor-Realease está formado por cuatro subdirectorios:

- bin: contiene los ejecutables accesibles para los usuarios.
- sbin: contiene los ejecutables accesibles únicamente para el administrador.
- etc: contiene los ficheros de configuración y ejemplos.
- lib: contiene librerias.

```
*********************************************************************
 STEP 4: How and where should Condor send email if things go wrong?
*********************************************************************

If something goes wrong with Condor, who should get email about it?
[condor@ghadir2.dacya.ucm.es]

What is the full path to a mail program that understands "-s" means
you want to specify a subject? [/bin/mail]

Using /bin/mail to send email to condor@ghadir2.dacya.ucm.es

Press enter to continue.
```

En este paso se le indica a quien debe mandar mails sobre errores, y que programa debe usar.
En nuestro caso, los mails se los manda al usuario condor de esa máquina, y utiliza el programa /bin/mail.

```
*********************************************************************
        STEP 5: Filesystem and UID domains.
*********************************************************************

To correctly run all jobs in your pool, including ones that aren't
relinked for Condor, you must tell Condor if you have a shared
filesystem, and if so, what machines share it.

Please read the "Configuring Condor" section of the Administrator's
manual (in particular, the section "Shared Filesystem Config File
Entries")
for a complete explaination of these (and other, related) settings.

Do all of the machines in your pool from your domain ("dacya.ucm.es")
share a common filesystem? [no]

Configuring each machine to be in its own filesystem domain.

Do all of the users across all the machines in your domain have a
unique UID (in other words, do they all share a common passwd file)?
[no]

Configuring each machine to be in its own uid domain.

Press enter to continue.
```

Le indicamos que las máquinas de nuestro dominio no comparten un mismo sistema de ficheros, y que cada máquina tiene su propio uid.

```
*******************************************************************
        STEP 6: Java Universe support in Condor.
*******************************************************************

Enable Java Universe support? [yes]

I wasn't able to find a valid JVM.

Please enter the full path to the JVM, or "none" to leave
unconfigured:


You entered:
That path doesn't start with an '/'.
Please try again.

Please enter the full path to the JVM, or "none" to leave
unconfigured:
none
OK, Java Universe will be left unconfigured.

Press enter to continue.
```

Nos pregunta si queremos configurar java, y le indicamos que si, pero como en ese momento no teníamos instalada la máquina virtual de java, al final no lo pudimos configurar. Tras la instalación se configuró manualmente.

```
*******************************************************************
        STEP 7: Where should public programs be installed?
*******************************************************************

The Condor binaries and scripts are already installed in:
        /home/condor/Condor_Release/bin

If you want, I can create some soft links from a directory that is
already in the default PATH to point to these binaries, so that
Condor users do not have to change their PATH.  Alternatively, I can
leave them where they are and Condor users will have to add
/home/condor/Condor_Release/bin to their PATH or explicitly use a
full pathname to access the Condor tools.

Shall I create links in some other directory? [yes]

Where should I install these files?
[/home/condor/bin]
/home/condor/bin isn't in your PATH, do you still want to use it?
[no] n

Where should I install these files?
[/home/condor/bin]
/home/condor/bin isn't in your PATH, do you still want to use it?
[no] y
/home/condor/bin is not a directory, should I create it? [yes]
```

```
Be sure to add /home/condor/bin to your PATH when this is done
so you can access the files I install.

Press enter to continue.
```

Este paso lo que hace es crear el directorio bin en el directorio condor con los ejecutables para los usuarios, de esta manera distinguimos al administrador del resto de los usuarios (para que un usuario normal no pueda ejecutar tareas propias del administrador).

Nos avisa de que /home/condor/bin no está en el PATH y nos pregunta si aun así queremos crear esos enlaces. Le indicamos que sí, y tras la instalación incluimos esa ruta manualmente en el PATH.

```
*********************************************************************
        STEP 8: What machine will be your central manager?
*********************************************************************

What is the full hostname of the central manager?
[ghadir2.dacya.ucm.es]

Your central manager will be on the local machine.

Press enter to continue.
```

Nos pregunta que máquina va a ser la que se comporte como central manager, y como estamos instalando el central manager le indicamos que nosotros mismos vamos a ser el central-manager.

```
*********************************************************************
        STEP 9: Where will the "local directory" go?
*********************************************************************

Condor will need to create a few directories for its own use

You have a "condor" user on this machine.  Do you want to put all the
Condor directories in /home/condor? [yes]

Creating all necessary Condor directories ... done.

Press enter to continue.
```

Nos pregunta donde queremos crear unos directorios necesarios para el funcionamiento de condor. Como tenemos un usuario condor en la máquina nos pregunta si los queremos crear en el home del usuario condor, le indicamos que si. Estos directorios que se crean son:

- spool
- log
- execute

En general sirven para registrar los eventos que se producen (ficheros de log).

```
*********************************************************************
    STEP 10: Where will the local (machine-specific) config files go?
*********************************************************************
```

```
Condor allows you to have a machine-specific config file that
overrides settings in the global config file.

You must specify a machine-specific config file.

Should I put a "condor_config.local" file in /home/condor?
[yes]
Creating config files in "/home/condor" ... done.

Configuring global condor config file ... done.
Created /home/condor/Condor_Release/etc/condor_config.

Press enter to continue.

Setting up ghadir2.dacya.ucm.es as your central manager

What name would you like to use for this pool?  This should be a
short description (20 characters or so) that describes your site. For
example, the name for the UW-Madison Computer Science Condor Pool is:
"UW-Madison CS".  This value is stored in your central manager's
local config file as "COLLECTOR_NAME", if you decide to change it
later.  (This shouldn't include any " marks).
GHADIR_POOL

Setting up central manager config file
/home/condor/condor_config.local ... done.

Press enter to continue.
```

Nos pregunta si queremos que se cree el fichero de configuración local en el home del usuario condor, que si existe otro lo sobrescribirá, le indicamos que si porque como es la primera vez que se instala no hay problema de que sobrescriba nada. Nos pide que introduzcamos el nombre del pool, y le llamamos GHADIR_POOL.

```
*********************************************************************
        STEP 11: How do you want Condor to find its config file?
*********************************************************************

Condor searches a few locations to find it main config file. The
first place is the envionment variable CONDOR_CONFIG. The second
place it searches is /etc/condor/condor_config, and the third place
is ~condor/condor_config.

Should I put in a soft link from /home/condor/condor_config to
/home/condor/Condor_Release/etc/condor_config [yes]

Installing links for public binaries into /home/condor/bin ... done.

Press enter to continue.
```

Nos pregunta si queremos hacer un enlace simbólico de /home/condor/condor_config a /home/condor/Condor_Release/etc/condor_config, le indicamos que si.

```
*******************************************************************
Condor has been fully installed on this machine.
*******************************************************************

/home/condor/Condor_Release/sbin contains various administrative
tools.If you are going to administer Condor, you should probably
place that directory in your PATH.

To start Condor on any machine, just execute:
/home/condor/Condor_Release/sbin/condor_master

Since this is your central manager, you should start Condor here
first.

Press enter to continue.

You should probably setup your machines to start Condor automatically
at boot time.  If your machine uses System-V style init scripts, look
in /home/condor/Condor_Release/etc/examples/condor.boot
for a script that you can use to start and stop Condor.

Please read the "Condor is installed... now what?" section of the
INSTALL file for things you should do before and after starting the
Condor daemons. In particular, you might want to set up host/ip
access security. See the Adminstrator's Manual for details.
[condor@ghadir2 condor-6.4.7]$
```

6.1.1.3. Instalación del Cliente

Los pasos de la instalación son similares a los del central-manager, solo cambia el paso 8 donde le indicamos que el central-manager es ghadir2.dacya.ucm.es.

6.1.2. INSTALACIÓN DE CONDOR EN WINDOWS

La instalación tanto del central manager como de los clientes sigue los mismos pasos que en la instalación para Linux. Lo único que cambia es que mientras Linux muestra los pasos de instalación por consola, Windows utiliza un entorno de ventanas más agradable. Por lo demás, son exactamente iguales.

6.1.3. INSTALACIÓN DE JAVA

Es necesario instalar java, ya que el universo que se va a utilizar en los ficheros submit es el universo java, y el .class que se manda en el fichero de submit pertenece a una clase java que será la que se ejecute con java en la máquina que Condor haya elegido para ejecutar el trabajo.

Para instalar java en Linux seguimos los siguientes pasos:

1. Creamos el directorio JDK_Source donde copiamos los archivos fuente de jdk y la documentación.
2. Descomprimimos estos ficheros en el directorio JDK_Release.
3. Acualizamos el fichero de configuración de condor (/home/condor/Condor_Release/etc/condor_config) donde modificamos el parámetro JAVA (que inicialmente estaba vacío) indicándole donde está el ejecutable de java. La línea resultantes será algo así como:
   JAVA = /home/condor/JDK_Release/jdk1.2.2/bin/java

La instalación de java en Windows es igual que la instalación de cualquier otra aplicación en un entorno Windows.

6.1.4. CONFIGURACIÓN DE LAS VARIABLES DE ENTORNO NECESARIAS EN LINUX

Hay que configurar la variable PATH y añadir una nueva variable de entorno llamada CONDOR_CONFIG.

6.1.4.1. Configuración de la variable PATH

Se añade al PATH la ruta a los enlaces de los comandos de condor para poder ejecutar esos comandos sin tener que situarnos en la carpeta donde se encuentran los enlaces, y así poder utilizar condor de una forma más rápida; y se añade también una ruta a los enlaces de los comandos de java por la misma razón.

Como la shell utilizada es tcsh, para configurar esta variable se ha tenido que crear el fichero .cshrc en el home de condor (porque no existía dicho fichero, si existiera solo habría de modificarlo). Este fichero es el profile para la shell tcsh.

A continuación se ha añadido la siguiente línea al fichero:

```
set path=($path /home/condor/bin /home/condor/JDK_Release/jdk1.2.2/bin)
```

De esta manera indicamos que la variable de entorno PATH tiene las rutas que tenía antes ($PATH), más las dos que se detallan a continuación.

6.1.4.2. Configuración de la variable CONDOR_CONFIG

Es obligatorio incluir esta variable de entorno para el correcto funcionamiento de condor, ya que indica la ruta donde Condor buscará el fichero de configuración.

Para añadir una variable de entorno, hay que modificar el fichero .cshrc incluyendo una línea que añada la variable e indique su valor. La línea a añadir en nuestro caso será:

```
setenv CONDOR_CONFIG /home/condor/Condor_Release/etc/condor_config
```

Esta línea añade la variable CONDOR_CONFIG al conjunto de variables de entorno y establece su valor a /home/condor/Condor_Release/etc/condor_config.

6.1.5. AJUSTES DEL CENTRAL MANAGER EN LINUX

En la máquina que va a actuar como central manager, debemos permitir al usuario condor (ya que éste se comportará como administrador) realizar algunas acciones de administración, tales como poner en funcionamiento el central manager, y apagarlo. Para ello, debemos incluir algunos enlaces de estas acciones que están en la carpeta /home/condor/Condor_Release/sbin (carpeta de administrador) a la carpeta /home/condor/bin, donde están los enlaces que puede ejecutar directamente el usuario condor.

Se añaden los siguientes enlaces:

1. Enlace que permite poner en funcionamiento el condor_master:

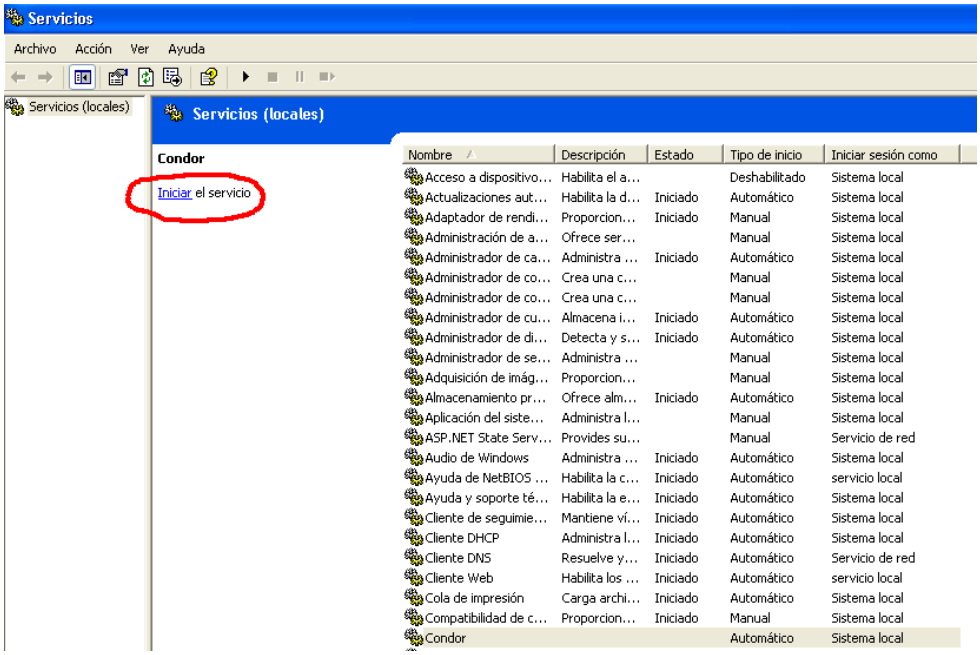
```
ln -s /home/condor/Condor_Release/sbin/condor_master home/condor/bin/condor_master
```

2. Enlace que permite finalizar el condor_master y clientes.

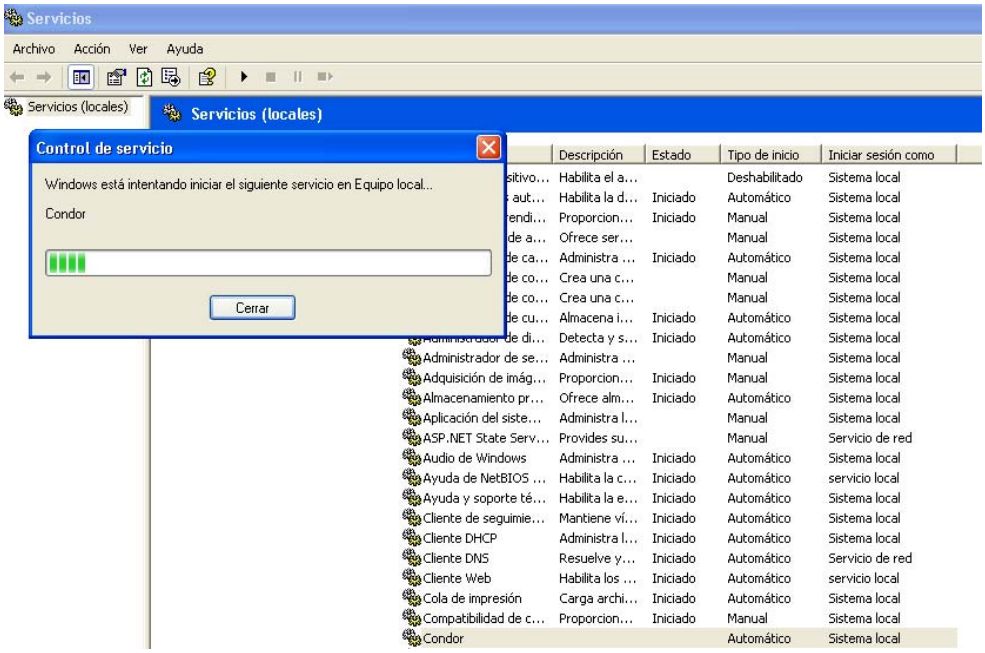
```
ln -s /home/condor/Condor_Release/sbin/condor_off    home/condor/bin/condor_off
```

6.1.6. CONFIGURACIONES Y AJUSTES EN WINDOWS

En Windows no ha sido necesario realizar ninguna configuración ni ajuste, ya que la instalación realiza todas las configuraciones y ajustes necesarios.
Hay que destacar que Condor en Windows se trata como un servicio. Para iniciarlo, nos introducimos en el panel de control → Herramientas Administrativas → Servicios, y ahí debe aparecer un servicio llamado Condor, lo seleccionamos y pulsamos Iniciar el servicio:



Servicio iniciándose:



6.1.7. MONTAJE DEL NFS EN LINUX

Aunque inicialmente montamos un NFS (Sistema de Ficheros Compartido) porque nos pareció interesante que todos los Jobs, Log y Resultados queden accesibles por parte de todos los participantes del Grid, mas tarde, exactamente cuando nos enfrentamos con el problema de cómo el cargador iba a proporcionarle a la aplicación que está en contacto con la FPGA el bitmap y los datos de configuración, nos dimos cuenta de que la única solución admisible de comunicación entre esas dos aplicaciones era a través de un NFS.

Decimos que es la única solución admisible ya que la otra solución más evidente (utilización de sockets para realizar la comunicación) haría que la elección que se tomó de elegir Condor perdiera todo su sentido, ya que si se utilizan sockets para realizar la comunicación entre máquinas, muy poco habría que añadirle para hacer la aplicación a medida que queremos sin utilizar Condor.

Pasos a seguir para la instalacion de NFS sobre una maquina Linux Red Hat 9:

1. Se comprueba que estén instalados nfs-utils y portmap en la máquina Ghadir2 con la siguiente línea de comando:

   ```
   rpm -q nfs-utils portmap
   ```

   Lo cual debe de regresar algo como lo siguiente:

   ```
   nfs-utils-0.3.1-13.7.2.1
   portmap-4.0-38
   ```

2. Se configura la seguridad.

Lo siguiente será configurar un nivel de seguridad para portmap en la máquina Ghadir2. Esto se consigue editando los ficheros /etc/hosts.allow y /etc/hosts.deny de dicha máquina. Debemos especificar que direcciones IP o rango de direcciones IP pueden acceder a los servicios de portmap y quienes no pueden hacerlo. Podemos entonces determinar en /etc/hosts.allow como rango de direcciones IP permitidas los siguientes:

```
portmap:192.168.1.0/255.255.255.0
```

Esto corresponde a la dirección IP de la red completa y la máscara de la sub-red. Adicionalmente podemos especificar direcciones IP individuales sin necesidad de establecer una máscara. Esto es de utilidad cuando se desea compartir volúmenes con otras máquinas en otras redes a través de Internet. Ejemplo:

```
portmap:192.168.1.0/255.255.255.0
portmap:192.168.20.25
portmap:192.168.30.2
portmap:216.200.152.96
portmap:148.240.28.171
```

En nuestro caso pondríamos portmap: 147.96.81.0/255.255.255.0 para que puedan acceder todas las máquinas de esa red.

Una vez determinado que direcciones IP pueden acceder a portmap, solo resta determinar quienes no pueden hacerlo. Evidentemente nos referimos al resto del mundo, y esto se hace agregando la siguiente línea en /etc/hosts.deny:

```
portmap:ALL
```

Es importante destacar que la línea anterior es indispensable y necesaria si quiere tener un nivel de seguridad decente. De manera predeterminada las versiones más recientes de nfs-utils no permitirán iniciar el servicio si esta línea no se encuentra presente en /etc/hosts.deny.

3. Una vez configurado portmap, reiniciamos el servicio de portmap en Ghadir2:

```
/sbin/service portmap restart
```

4. Si se tiene un DNS, hay que dar de alta las direcciones IP asociadas a un nombre, si no se tiene un DNS editar /etc/hosts y agregue las direcciones IP asociadas con un nombre. Esto nos servirá como listas de control de accesos.

   Ejemplo del fichero /etc/hosts:

```
127.0.0.1       localhost.localdomain   localhost
192.168.1.254   servidor.mi-red-local.org servidor
192.168.1.2     algun_nombre.mi-red-local.org    algun_nombre
192.168.1.3     otro_nombre.mi-red-local.org     otro_nombre
192.168.1.4     otro_nombre_mas.mi-red-local.org otro_nombre_mas
192.168.1.5     como_se_llame.mi-red-local.org    como_se_llame
192.168.1.6     como_sea.mi-red-local.org como_sea
192.168.1.7     lo_que_sea.mi-red-local.org       lo_que_sea
```

En nuestro caso como no tenemos DNS tenemos que editar el fichero /etc/hosts. La máquina que estamos configurando es ghadir2, y la máquina a la que quiero tener acceso es ghadir1, así que hay que introducir en /etc/host la línea:

```
147.96.81.67      Ghadir1.dacya.ucm.es    Ghadir1
```

5. Se comparte un volumen NFS.

Procederemos a determinar que directorio se va a compartir. Se puede crear también uno nuevo con el siguiente comando:

```
mkdir -p /home/condor/compartido
```

Una vez hecho esto, necesitaremos establecer que directorios en el sistema serán compartidos con el resto de las máquinas de la red, o bien a que máquinas, de acuerdo al DNS o /etc/hosts se permitirá el acceso. Esto deberemos agregarlos en /etc/exports determinado con que máquinas y en que modo lo haremos. Se puede especificar una dirección IP o bien nombre de alguna máquina, o bien un patrón común con comodín para definir que máquinas pueden acceder. De tal modo podemos utilizar el siguiente ejemplo (la separación de espacios se hace con un tabulador):

```
/home/condor/compartido         *.dacya.ucm.es(ro,sync)
```

En el ejemplo anterior se esta definiendo que se compartirá /home/condor/compartido a todas las máquinas cuyo nombre, de acuerdo al DNS o /etc/hosts, tiene como patrón común dacya.ucm.es, en modo de lectura escritura. Se utilizó un asterisco (*) como comodín, seguido de un punto y el nombre del dominio. Esto permitirá que Ghadir1 pueda acceder al volumen /home/condor/compartido en modo solo lectura. Si tuviéramos más máquinas pertenecientes a la red dacya.ucm.es en el fichero /etc/hosts, estas también podrían acceder al volumen /home/condor/compartido. Si queremos que el acceso a este directorio sea en modo de lectura y escritura, cambiamos (ro) por (rw):

```
/home/condor/compartido         *.dacya.ucm.es (rw,sync)
```

6. Se inicia o reinicia el servicio nfs en Ghadir2. Se pueden utilizar cualquiera de las dos líneas dependiendo el caso:

```
/sbin/service nfs start
/sbin/service nfs restart
```

Para asegurarnos de que el servicio de nfs esté habilitado la siguiente vez que se encienda el equipo, debemos ejecutar lo siguiente:

```
/sbin/chkconfig --level 345 nfs on
```

El comando anterior hace que se habilite nfs en los niveles de corrida 3, 4 y 5.

Como medida de seguridad adicional, si tiene un contrafuegos o firewall implementado, cierre, para todo aquello que no sea parte de su red local, los puertos tcp y udp 2049, ya que estos son utilizados por NFS para escuchar peticiones.

Estos pasos los hemos realizado sobre la maquina Ghadir2.dacya.ucm.es, con ip 147.96.81.68, en el directorio /home/condor/compartido.

7. Configuración de las máquinas cliente.

En nuestro caso, la única máquina que tenemos que configurar es Ghadir1.

Es necesario que las máquinas clientes se encuentren definidas en el DNS o en el fichero /etc/hosts del servidor, en nuestro caso eso se cumple pues la hemos introducido en el paso 4.

Como root, en el equipo cliente, se ejecuta el siguiente comando para consultar los volúmenes exportados (-e) a través de NFS por un servidor en particular:

```
showmount -e 147.96.81.68
```

Lo anterior mostrará una lista con los nombres y rutas exactas a utilizar.
Ejemplo:

```
Export list for 147.96.81.68:
/home/condor/compartido          147.96.81.0/255.255.255.0
```

A continuación creamos, como root, desde la máquina Ghadir1 un punto de montaje:

```
mkdir /home/condor/compartido
```

Y para proceder a montar el volumen remoto, utilizaremos la siguiente línea de comando si en el fichero /etc/hosts tenemos asociado el nombre de la máquina Ghadir2 con su IP :

```
mount Ghadir2.dacya.ucm.es: /home/condor/compartido  /home/condor/compartido
```

Si en el fichero /etc/hosts no tenemos asociado el nombre de la máquina Ghadir2 con su IP se escribirá la siguiente línea de comando:

```
mount -t nfs 147.96.81.68: /home/condor/compartido  /home/condor/compartido
```

Podremos acceder entonces a dicho volumen remoto con solo cambiar al directorio local definido como punto de montaje.

Cabe destacar que en Linux cada vez que se arranque la máquina deberemos hacer el montaje del volumen remoto escribiendo una de los dos comandos anteriores según proceda, si se quiere que el volumen se monte automáticamente, simplemente habría que hacer los cambios necesarios en la tabla /etc/fstab.

Estos pasos se han realizado sobre la maquina cliente en este caso ghadir1.dacya.ucm.es con ip 147.96.81.67 y sobre el directorio /home/condor/compartido

Finalmente destacar los problemas surguidos y superados como fueron aspectos de seguridad y configuracion del firware de la maquina en nuestro caso “iptables” para permitir el acceso a dichos directorios.

6.1.8. COMPARTICIÓN DE UNA CARPETA EN WINDOWS

La compartición de ficheros en Windows se hace de forma automática, tan solo hay que elegir la carpeta que queremos compartir, hacer clic sobre ella con el botón derecho y seleccionar compartir. Además en Windows no es necesario que las máquinas clientes monten ese volumen se puede acceder directamente escribiendo //Ghadir2/ruta_fichero_compartido.

6.2. FICHEROS DE CONFIGURACIÓN

Hasta este momento nos hemos dedicado a comprender como se instala, configura y utiliza la herramienta que vamos a utilizar (Condor), este es el momento ahora de ponernos a ver si es factible modificar su comportamiento para admitir la posibilidad de recursos hardware, para ello primero investigaremos lo que es el método utilizado por Condor para comunicar la información, esto es el ClassAds un mecanismo por el cual cada maquina del pool conoce los recursos que otra maquina expone para su uso visto en el [Apartado 5.2.2], de esta manera pretendemos añadir un nuevo recurso, que represente a nuestra tarjeta FPGA (bool FPGA_STATUS en su primera aproximación, que representa si la FPGA esta Libre).

Lo primero es como se define un nuevo atributo en una maquina, pues esta parte es aparentemente la mas fácil, basta con añadirla en un principio al fichero de configuración local (C:\Condor\etc\ o \home\Condor_Release\etc\), como un nuevo atributo y asignarle un valor, pero esto no es suficiente, pues ¿Como se entera Condor de la existencia de este atributo? Para solucionar esto necesitamos definir dicha variable en un macro (son las expresiones que utiliza Condor para definir variables en el ClassAd) denominado por Condor como STRTD_EXPRS, en la cual se definen los atributos definidos por el usuario de tal manera que el demonio startd se entere de su existencia y los comunique mediante ClassAd, al resto de maquinas del pool, de esta manera definimos el nuevo atributo FPGA_STATUS, asignándolo a TRUE inicialmente, con esto y mediante el comando condor_status -l podemos observar (tras reiniciar Condor o ejecutar condor_reconfig) la existencia de dicho atributo y su valor de TRUE, con esto ya podemos enviar tareas dependientes de dicho atributo y en función de su valor, para nuestro propósito global necesitaremos además de esta variable que indica el estado de la FPGA, otra que nos indique si el ordenador dispone o no de un recurso de HW reconfigurable, esta será HasFPGA, que a su vez también será booleana.

Las Variables introducidas quedan de la siguiente manera:

```
HasFPGA = TRUE
FPGA_STATUS = TRUE
STARD_EXPRESS = HasFPGA, FPGA_STATUS
```

A pesar de esto necesitamos crear una aplicación, que se detalla en el [Apartado 6.3.1] con la cual poder cambiar de una forma fácil el contenido de estos atributos, el mecanismo es fácil, consiste en modificar el valor de dicho atributo, en el fichero de configuración y forzar a Condor a volver a leerlo con la llamada a condor_reconfig, de este modo se consigue activar el mecanismo de ClassAd y en unos segundo el valor modificado de la variable es visible desde cualquier maquina del pool.

Con la creación de las variables necesarias para implementar cualquier tipo de gestión, podemos definir multitud de recursos diferentes, ya sean Hardware o Software, y aprovechando el mecanismo de ClassAd, tenemos una forma fácil de entender y utilizar y con mucha flexibilidad para nuestros propósitos.

Estos cambios son equivalentes tanto en Windows como en Linux, pues los ficheros de configuración son los mismos en ambos S.O.

6.3. APLICACIONES REALIZADAS

6.3.1. SCRIPT DE CAMBIO DE VARIABLE

Para ejecutar una tarea en una FPGA, se realiza un fichero de submit de esa tarea indicando en los requerimientos que la máquina tenga FPGA y que debe tener suficiente FPGA para ejecutarla. La manera en que Condor sabe si la máquina tiene tarjeta FPGA y si tiene suficiente FPGA para ejecutar la tarea es consultando dos variables que se encuentran en el fichero de configuración de Condor (condor_config) donde se guarda la información sobre el estado en el que se encuentran los recursos de la máquina, estas variables son: HasFPGA para saber si la máquina tiene tarjeta FPGA o no, y FPGA_STATUS para saber la cantidad de espacio en la FPGA que tiene libre; así pues, Condor consultando esas variables envía la tarea a una máquina cuya variable HasFPGA le indique que la máquina tiene tarjeta FPGA y cuya variable FPGA_STATUS le indique que tiene suficiente FPGA para ejecutarla.

La línea que hay que incluir en el fichero de submit para que envíe la tarea a la máquina apropiada es:

```
Requirements      = HasFPGA==TRUE && FPGA_STATUS==TRUE.
```

Condor se encarga de buscar la máquina donde ejecutar la tarea, pero una vez asigna la tarea a una máquina no actualiza la variable FPGA_STATUS para indicar que hay un espacio de la FPGA que está ocupado. Por ello, hemos tenido que realizar un pequeño script que se encargue de actualizar esa variable.

Este script que actualiza la variable FPGA_STATUS se ejecutará justo antes de que se empiece a ejecutar la tarea en la FPGA para indicar que hay un espacio ocupado en la FPGA, y tras acabar de ejecutar la tarea para indicar que el espacio que estaba ocupando la tarea en la FPGA vuelve a estar libre.

La decisión que se ha tomado respecto a la gestión de la FPGA, cuantas tareas puede ejecutar en un mismo instante y cual es la forma de las tareas en la FPGA, es realizar una gestión estática, esto es que la FPGA tan sólo tiene dos estados: libre u ocupada y sólo puede ejecutar una tarea cada vez [Apartado 3.2]. La modificación de la representación de la FPGA para que pueda ejecutar más de una tarea en un instante dado se abordará en el [Apartado 6.3].

Como sólo tenemos que representar dos estados, la variable FPGA_STATUS puede ser perfectamente una variable booleana, así pues, cuando se ponga a TRUE indicará que la FPGA está libre y cuando se ponga a FALSE indicará que la FPGA está ocupada.

6.3.1.1. Linux

Se ha realizado un script en Linux que se llama condor_set_var, y su contenido es el siguiente:

```
#! /bin/tcsh
awk '{if($1 ~ "FPGA_STATUS") { print "FPGA_STATUS = " argumento }
else              print              $0}'              argumento=$1
"/home/condor/Condor_Release/etc/condor_config"                    >
"/home/condor/Condor_Release/etc/tmp"
mv                              "/home/condor/Condor_Release/etc/tmp"
"/home/condor/Condor_Release/etc/condor_config"

/home/condor/Condor_Release/sbin/condor_reconfig
```

Lo primero que hacemos es ejecutar un pequeño programa en el lenguaje awk [(1)] que es soportado por Linux:

```
awk '{if($1 ~ "FPGA_STATUS") { print "FPGA_STATUS = " argumento }
else print $0}' argumento=$1
"/home/condor/Condor_Release/etc/condor_config"
```

Este programa en awk recibe un argumento $1 que lo guardamos en 'argumento'. A awk hay que indicarle el fichero que tiene que leer, el fichero que tiene que tratar es el fichero de configuración de Condor cuya ruta es: /home/condor/Condor_Release/etc/condor_config. Lo que hace awk es leer línea a línea el fichero indicado y para cada línea lo que hace es:

1. Guarda la línea entera en $0.
2. Guarda cada palabra de la línea en una variable distinta empezando por $1. Por ejemplo, si una línea del fichero es: "Mi nombre es Juana", tendríamos que $0 = Mi nombre es Juana, $1 = Mi, $2=nombre, $3=es, $4=Juana.
3. Ejecuta el programa que hay dentro de las comillas simples.

En nuestro caso el progama que debe de ejecutar para cada línea es:

```
'{if($1 ~ "FPGA_STATUS") { print "FPGA_STATUS = " argumento } else print $0}'
```

Lo que hace es comparar la primera palabra de la línea con FPGA_STATUS, si coincide (($1 ~ "FPGA_STATUS") lo que hace es imprimir en vez de esa línea la línea "FPGA_STATUS = " argumento; y en el caso de que no coincida escribimos la línea tal cual está (else print $0)

Este programa en awk lo que hace es mostrar por pantalla el contenido del fichero de configuración con el valor de la variable FPGA_STATUS ya actualizado al valor que le hayamos pasado por argumento.

Lo que hacemos a continuación es redirigir la salida del programa a un fichero para que en vez de mostrarlo por pantalla lo mande a un fichero temporal que lo creará en la ruta: "/home/condor/Condor_Release/etc/tmp".

A continuación se renombra el fichero para que pase a llamarse condor_config de la siguiente manera:

```
mv "/home/condor/Condor_Release/etc/tmp"
"/home/condor/Condor_Release/etc/condor_config".
```

Por último, hay que obligar a Condor a que vuelva a leer el fichero de configuración porque sino no tendrían efecto los cambios realizados, para ello se ejecuta el siguiente comando:

```
/home/condor/Condor_Release/sbin/condor_reconfig.
```

Así pues, para utilizar este script para indicar que la FPGA está libre se hará: condor_set_var TRUE; y para indicar que está ocupada se hará: condor_set_var FALSE;

(1). Awk es un lenguaje de búsqueda y procesamiento de patrones. Esto quiere decir que awk es capaz de buscar un patrón dentro de un archivo (al igual que grep) y tratarlo según unas operaciones determinadas. Awk es capaz de procesar un archivo con datos organizados por columnas y generar nuevas columnas con los valores resultantes de realizar ciertos cálculos u operaciones.

6.3.1.2. Windows

Para Windows se ha realizado un programa en Java que se llama condor_set_var. Este programa sirve para cambiar cualquier variable de valor, y su utilización es ejecutar condor_set_var pasándole como primer argumento la variable que queremos cambiar, y como segundo argumento el valor que le queremos dar.

Si queremos indicar que la FPGA está libre ejecutaremos condor_set_var pasándole como primer argumento FPGA_STATUS y como segundo TRUE; si por el contrario, queremos indicar que está ocupada, se ejecutará poniendo como primer argumento FPGA_STATUS y como segundo FALSE.

Este programa sirve tanto para Windows como para Linux. El código del programa es el siguiente:

```
import java.io.*;
import java.lang.*;

public class condor_set_var{

  public static void main(String[] args){

  String str;
  String lineaModificada;

  try{
        File fiOrig = new File("c:\\Condor\\condor_config");
        File fiTemp = new java.io.File("c:\\Condor\\condor_config.new");
        BufferedReader entrada = new BufferedReader(new FileReader(fiOrig));
        BufferedWriter salida = new BufferedWriter(new FileWriter(fiTemp));
        String linea = entrada.readLine();
        while (linea != null ){
        if ( linea.startsWith(args[0]) ){
              salida.write(args[0] +" = " + args[1]
              +System.getProperty("line.separator"));
        }
        else
              salida.write(linea +System.getProperty("line.separator"));
              linea = entrada.readLine();
        }// fin del while

        entrada.close();
        salida.flush();
        salida.close();
        fiOrig.delete();
        if (! fiTemp.renameTo(fiOrig))
              System.out.println("Se ha renombrado mal");



  }
  catch (Throwable e) {
        System.out.println("Error");
  }
  }
}
```

El funcionamiento de este programa es el que se detalla a continuación:

Se va a utilizar un fichero auxiliar donde se escribirá todo el fichero condor_config tal cual está exceptuando la línea correspondiente a la variable que queremos modificar donde se escribirá argumento1 = argumento2.

Para ello se realiza un bucle que lee todo el fichero condor_config línea y línea. Para realizar la lectura del fichero se utiliza un objeto que se llama BufferedReader que tiene un método que se llama readLine() que nos permite leer la siguiente línea del fichero. Para cada línea leída se comprueba si hace referencia a la variable que queremos modificar, para ello se mira si el comienzo de la línea coincide con lo que le hemos pasado en el primer argumento: if (linea.startsWith(args[0])). Si no coincide se escribe la línea tal cual está en un fichero auxiliar ("c:\Condor\condor_config.new") utilizando el objeto BufferedWriter; y si coincide se escribe la línea args[0] +" = " + args[1] +System.getProperty("line.separator") que imprime argumento1 = argumento2 con un salto de línea a continuación.

Cuando hemos procesado todo el fichero renombramos el fichero temporal para que pase a llamarse condor_config (fiTemp.renameTo(fiOrig)).

6.3.2. CARGADOR

Una vez Condor haya elegido la máquina donde se va a ejecutar la tarea, tenemos que comunicarnos con la aplicación que se encarga de ejecutar la tarea en la FPGA e indicarle cual es el bitmap que debe de ejecutar y cuales son los datos de configuración. Para ello se ha realizado un programa en Java, el cargador, que será el que se envíe en el fichero de submit y el que se ejecutará en la máquina que Condor haya elegido. En el fichero de submit aparecerá lo siguiente:

Executable = TCargador.class, que le dirá que el ejecutable es la clase del cargador originada tras compilarlo.

Para establecer la comunicación entre las dos aplicaciones se ha optado por la utilización de un socket. Así pues, la aplicación que ejecuta las tareas tendrá un socket abierto a la espera de conexiones, y el cargador lo que hará será conectarse a ese socket.

Una vez ya tengamos establecida la comunicación entre las dos aplicaciones, lo siguiente será proporcionarle el bitmap y los datos de configuración. Para ello se ha utilizado un sistema compartido de ficheros entre todas las máquinas del pool para no tener que enviar esos datos de una máquina a otra. Se ha elegido esta opción ya que si las máquinas están en red lo más normal es que tengan alguna carpeta con ficheros compartidos. De esta manera el cargador no deberá enviar de una máquina a otra el bitmap ni los datos de configuración, simplemente deberá indicarle las rutas de esos dos ficheros, que se encontrarán en una carpeta compartida por todas las máquinas. Así pues, el cargador recibirá dos argumentos, que serán las dos rutas a los ficheros que contienen el bitmap y los datos de configuración. En el fichero de submit aparecerá algo como lo siguiente:

Arguments = TCargador /home/condor/compartido/bitmap /home/condor/compartido/datosConf.txt.

Si lo que se quiere es finalizar la aplicación que ejecuta las tareas en la FPGA se realizará lo mismo que antes, excepto que no le pasaremos las dos rutas como parámetros al cargador, sino que le mandaremos un solo argumento que será FINALIZAR, y esto ocasionará la parada de la aplicación.

El código del cargador es el mismo tanto en Linux como en Windows, y es el siguiente:

```
/**
 * <p>Title: Cargador-Simulador v1.0</p>
 * <p>Description: Recibe un Job y lo envia al Simulador</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: GHADIR</p>
 * @author Ignacio JosÃ© Horrillo Sancho
```

```
 * <p> GermÃ¡n Hurtado MartÃ-n</p>
 * <p> Ana MarÃ-a VÃ©lez Villanueva </p>
 * @version 1.0
*/

  //Paquetes necesarios
  import javax.swing.JOptionPane;
  import java.io.*;
  import java.net.*;

public class TCargador {

  /**Puerto del Simulador mediante el cual ser va a realizar la conexión*/
  final static int SERVERPORT = 7777;
  /**Dirección IP del Simulador*/
  private String IP;
  /**Socket mediante el cual el Cargador se comunica con el Simulador*/
  private Socket sock;
  /**Canal de escritura con el Simulador*/
  private ObjectOutputStream oos;    //Para escribir
  /**Canal de lectura con el Simulador*/
  private ObjectInputStream ois;     //Para leer

  /**Constructor de la clase.
   * Abre el socket y los canales de lectura/escritura
  */
  public TCargador(String ip) throws IOException, UnknownHostException
  {
    IP = ip;
    sock = new Socket(InetAddress.getByName(IP),SERVERPORT);
    oos = new ObjectOutputStream(sock.getOutputStream());
    ois = new ObjectInputStream(sock.getInputStream());
  } // Fin TCargador()

  public static void main(String[] args) {
    try
    {
       String ip = "localhost
       TCargador cargador = new TCargador(ip);
 /*Le pasamos a traves del socket todos los datos necesarios es decir la
dir de la img y la configuracion del Job*/

       if(args[0].equals("FINALIZAR")){
          System.out.println(args[0]);
          cargador.oos.writeObject(args[0]);
       }
       else{
          int i=0;
          while (i<2){
            System.out.println(args[i]);
            cargador.oos.writeObject(args[i]);
            i++;
          }
       }
       cargador.oos.writeObject(null);
/*Esperamos un mensaje de finalizacion por parte del Simulador para darle
tiempo a recibir los mensajes antes de cerrar la comunicacion*/
       String fin = (String)cargador.ois.readObject();
       if (fin.equals("FIN")) System.out.println("FIN");
```

```
       else System.out.println("ERROR="+fin);
       /* Cierre del canal de I/O y el Socket*/
       cargador.oos.close();
       cargador.ois.close();
       cargador.sock.close();
    } //Fin try
    catch (Throwable e){
       System.out.println("Error" + e.getMessage());
       e.printStackTrace();
    } // Fin catch (Throwable e)
  }//Fin main(String[] args)
}//Fin class TCargador
```

Lo primero que hace el cargador es abrir un socket para comunicarse con la otra aplicación:
sock = new Socket(InetAddress.getByName(IP),SERVERPORT);

Donde la ip que se le manda será "localhost" ya que el socket servidor está en la misma máquina. A continuación abrimos dos flujos, uno de entrada y uno de salida para poder realizar la comunicación:

```
oos = new ObjectOutputStream(sock.getOutputStream());
ois = new ObjectInputStream(sock.getInputStream());
```

A continuación se comprueba si el primer argumento que se le ha pasado al cargador ha sido FINALIZAR o si por el contrario se le han mandado las dos rutas; en cualquier caso se envía a la otra aplicación mediante el flujo de salida o bien FINALIZAR o bien las dos rutas. Al final se le manda un mensaje nulo para que la otra aplicación sepa que ya no se van a enviar más datos:

```
cargador.oos.writeObject(null);
```

Para no cerrar el socket antes de que la otra aplicación haya leído todo, incluido el mensaje null, el cargador esperará hasta que se reciba por el flujo de lectura del socket un mensaje con el texto "fin" que le enviará la otra aplicación tras leer el mensaje null:

```
String fin = (String)cargador.ois.readObject();
if (fin.equals("FIN")) System.out.println("FIN");
```

Una vez leído ese mensaje, el cargador ya puede proceder a cerrar los flujos de entrada y salida del socket, y el socket, garantizándose que la otra aplicación ha recibido todo:

```
cargador.oos.close();
cargador.ois.close();
cargador.sock.close();
```

Así pues, un ejemplo de un fichero de submit que manda un bitmap para ejecutar que se llama /home/condor/compartido/animo, con unos datos de configuración guardados en /home/condor/compartido/animo2.txt sería el siguiente:

```
####################################
#                                  #
# Submit para el Cargador           #
# Fichero de Descripcion del Job   #
#                                  #
####################################

Executable  = TCargador.class
Requirements      = HasFPGA==TRUE && FPGA_STATUS==TRUE
Priority    = 0
```

```
Arguments   = TCargador /home/condor/compartido/animo
/home/condor/compartido/animo2.txt
Universe    = java
Output            = result/out/cargador.out
Log         = result/log/cargador.log
transfer_files    = ONEXIT
Queue
```

Para más información sobre los ficheros submit véase el [Apartado 5.3].

6.3.3. SIMULADOR

Se ha realizado una pequeña aplicación que simule ser la aplicación que ejecuta la tarea sobre la FPGA para poder comprobar el correcto funcionamiento del cargador.

Esta aplicación en vez de ejecutar la tarea sobre la FPGA lo que hace es mostrar un mensaje diciendo que se está ejecutando el trabajo, y para comprobar que la aplicación es capaz de encontrar los archivos en el sistema de ficheros compartido usamos ficheros de texto y editores de texto para mostrar el contenido que tienen los dos fichero especificados por las rutas que le pasa el cargador. Si estamos en un entorno Linux se muestran con la aplicación gedit, y si estamos en un entorno Windows se muestran con la aplicación notepad.

6.3.3.1. Linux

El código del simulador para Linux es el siguiente:

```
/**
 * <p>Title: Cargador-Simulador v1.1</p>
 * <p>Description: Recibe los datos enviados por el Cargador</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: GHADIR</p>
 * @author Ignacio JosÃ© Horrillo Sancho
 * <p> GermÃ¡n Hurtado MartÃn</p>
 * <p> Ana MarÃa VÃ©lez Villanueva </p>
 * @version 1.0
 */

//Paquetes necesarios
  import javax.swing.JOptionPane;
  import java.io.*;
  import java.net.*;


public class TSimulador implements Runnable{

  Thread hilo;
/**Puerto de escucha del Simulador*/
  final static int SERVERPORT = 7777;
  /**Socket de escucha*/
  private ServerSocket sSock;
  /**Socket que conecta al Cargador*/
  private Socket CargadorSocket;
  /**Canal de escritura con el Cargador*/
  private ObjectOutputStream oos;    //Para escribir
  /**Canal de lectura con el Cargador*/
  private ObjectInputStream ois;     //Para leer
```

```
  public TSimulador() throws IOException, UnknownHostException{
    try {
      hilo = new Thread(this);
      hilo.start();
    } catch (Exception e){
      e.printStackTrace();
    }
  }//Fin TSimulador

  public void run(){
    while (true){
        //System.out.println("Esperando");
      if (comprobarCargador()){
         System.out.println("Finalizado.");
         return;
      }
    }
  }

  public boolean comprobarCargador(){
  try{
      sSock = new ServerSocket(SERVERPORT);
      System.out.println("Esperando al cargador...");
    /*Preparamos el Socket del Simulador para esperar conexiones*/
     CargadorSocket = sSock.accept();
     /*Ejecutamos el Script para actualizar el FPGA_STATUS*/
     TCall llamada = new TCall();
     llamada.ejecutarProgramaWindows("condor_set_var FALSE");
/*Abrimos los canales de lectura/escritura para comunicarnos con el
Cargador*/
     oos = new ObjectOutputStream(CargadorSocket.getOutputStream());
     ois = new ObjectInputStream(CargadorSocket.getInputStream());

     /*Recibimos los mensajes del Cargador*/
     String msgs[] = new String[3];
     String msg;
     int i=0;
     do{
       msg = (String)ois.readObject();
       msgs[i]=msg;
       i++;
       System.out.println(msg);
     } while(msg!=null);
     /*Enviamos el mensaje de FIN*/
     oos.writeObject("FIN");
     /* Cierre del canal de I/O y el Socket*/
     oos.close();
     ois.close();
     CargadorSocket.close();
     sSock.close();
     /*Ejecutamos el Job*/
     if (msgs[0].equals("FINALIZAR")){
        llamada.ejecutarProgramaWindows("condor_set_var TRUE");
        return true;
     }
     else {
        System.out.println("Ejecutando Job...");
        llamada.ejecutarProgramaWindows("gedit "+msgs[0]);
```

```
        llamada.ejecutarProgramaWindows("gedit "+msgs[1]);
        /*Ejecutamos el Script para actualizar el FPGA_STATUS*/
        llamada.ejecutarProgramaWindows("condor_set_var TRUE");
        return false;
     }
  } // Fin try
  catch (Throwable e)
  {
     System.out.println("Error" + e.getMessage());
     e.printStackTrace();
     return true;
  } // Fin catch (Throwable e)
}

  public static void main(String[] args) {
   try
   {
     TSimulador simulador = new TSimulador();
  } // Fin try
  catch (Throwable e)
  {
     System.out.println("Error" + e.getMessage());
     e.printStackTrace();
  } // Fin catch (Throwable e)
  }//Fin main(String[] args)

}
```

El simulador va a ser un hilo que se esté ejecutando siempre hasta que se le mande una tarea especial que le haga finalizarse. Esa tarea especial consiste en un mensaje con el texto “FINALIZAR” en vez de recibir las dos rutas que indican el bitmap a ejecutar y los datos de configuración.

El simulador abre un server socket por el que espera conexiones (`CargadorSocket = sSock.accept();`), una vez recibida una conexión actualizamos el estado de la variable FPGA_STATUS para indicar que está ocupada, la modificación se hace aquí para evitar que otra tarea se adelante, para modificar esta variable se debe ejecutar condor_set_var FALSE, para ejecutarlo utilizaremos un método de un objeto de la clase TCall que se explicará más adelante.

A continuación abrimos unos flujos de entrada y salida para comunicarnos con el cargador:

```
oos = new ObjectOutputStream(CargadorSocket.getOutputStream());
ois = new ObjectInputStream(CargadorSocket.getInputStream());
```

Leemos los datos que nos vengan por el flujo de entrada hasta recibir un mensaje null que indica que ya no hay más datos que leer.

Se envía al cargador el mensaje de “fin” para que sepa que se han recibido todos los datos correctamente:

```
oos.writeObject("FIN");
```

Se comprueba si se ha recibido la tarea especial que hace finalizar el simulador, y en ese caso se actualiza la variable FPGA_STATUS para indicar que la FPGA está libre. Si por el contrario, se ha recibido una ruta de un bitmap a ejecutar y la ruta del fichero con los datos de configuración se imprime un mensaje que indica que el trabajo se está ejecutando (aquí habría que ejecutarlo realmente sobre la FPGA).

Una vez finalizado el trabajo, su vuelve a actualizar la variable FPGA_STATUS para indicar que vuelve a estar libre.

6.3.3.2. Windows

El código del simulador para Windows es el mismo que en Linux, lo único que cambia respecto al código en Linux es la forma en la que se ejecuta el comando que actualiza el valor de la variable FPGA_STATUS, en el caso de Windows al programa java condor_set_var hay que pasarle dos argumentos, el nombre de la variable y el nuevo valor. Además ese comando se tiene que ejecutar en la línea de comandos, por eso el comando completo a ejecutar es cmd /c condor_set_var FPGA_STATUS TRUE.

6.3.3.3. Ejecutor de Comandos: TCall

Se ha desarrollado una clase en Java llamada **TCall** que sirve para ejecutar comandos tanto en Windows como en Linux. Su código es el siguiente:

```
/**
 * <p>Title: Cargador-Simulador v1.0</p>
 * <p>Description: ejecuta una aplicación externa</p>
 * <p>Copyright: Copyright (c) 2004</p>
 * <p>Company: GHADIR</p>
 * @author Ignacio José© Horrillo Sancho
 * <p> Germán Hurtado Martín</p>
 * <p> Ana María Vélez Villanueva </p>
 * @version 1.0
 */

  /* Paquete para tratar excepciones de entrada/salida */
  import java.io.IOException;

public class TCall {

    private Runtime ejecutor;
    private Process proceso;
    private String comando;

  public TCall() {
    ejecutor = Runtime.getRuntime();
    proceso = null;
    comando = null;
  }
  void ejecutarProgramaWindows (String cmd) {
    try {
      comando = cmd;
      proceso = ejecutor.exec(cmd);
      // Esta linea es para que la aplicacion Java se quede en espera
      // hasta que la aplicacion Windows se cierre
      proceso.waitFor();
      System.out.println(cmd);
    }  //fin try
    catch(InterruptedException ie) {
      System.err.println("Error al ejecutar, comando = "+cmd);
      System.err.println("Excepcion capturada: "+ie);
    } //fin catch
    catch(IOException ioe) {
```

```
      System.err.println("Error al ejecutar, comando = "+cmd);
      System.err.println("Excepcion capturada: "+ioe);
    } //fin catch

    // Esta linea es para mostrar el codigo con el que termina el
proceso
    System.out.println("El comando "+cmd+" ha teminado con código
"+proceso.exitValue());
  }
}
```

Para ejecutar cualquier comando lo único que hay que hacer es crear un objeto de la clase descrita anteriormente (TCall) y llamar al método ejecutarProgramaWindows pasándole como argumento el comando que queremos ejecutar.

La clase TCall utiliza la clase Runtime que tiene un método llamado exec al que le pasas un comando y te devuelve un proceso (un objeto de la clase Process) asociado a ese comando. Una vez obtenido el proceso, se espera a que el proceso termine utilizando un método de la clase Process que se llama waitFor.

6.3.3.4. Programa de Arranque del Simulador: simulador_on

Se ha desarrollado un pequeño proceso por lotes para arrancar el simulador.

6.3.3.4.1 Linux

El código del programa es el siguiente:

```
java -cp /home/condor/compartido/ Tsimulador
```

Para que el programa funcione tiene que estar compilado el fichero donde se encuentra la clase Tsimulador, y la clase que se origina tras la compilación debe estar en la ruta: "/home/condor/compartido/".

Lo único que hace es ejecutar java con la opción -cp (o -classpath), en la que se indica el directorio en el que están las clases (/home/condor/compartido/), pasándole como argumento la clase a ejecutar, en nuestro caso Tsimulador.

6.3.3.4.2 Windows

El código del programa es el siguiente:

```
C:\jdk1.2.2\bin\java -cp c:\Condor\bin\ Tsimulador
```

Este programa es un .bat y lo que hace es ejecutar java que se encuentra en la ruta: "C:\jdk1.2.2\bin\" con la opción –cp para indicarle que las clases están en el directorio "c:\Condor\bin\", pasándole como argumento la clase a ejecutar: Tsimulador.

6.3.3.5. Programa de Parada del Simulador: simulador_off

Se ha desarrollado un pequeño programa en Java para parar el simulador. En ambos casos, tanto para Windows como para Linux, para parar el simulador lo que se hace es ejecutar un cargador con argumento igual a FINALIZAR, lo que hace que el cargador se comunique con el simulador y le indique que debe terminar.

6.3.3.5.1. Linux

El código del programa es el siguiente:

```
java -cp /home/condor/compartido/ TCargador FINALIZAR .
```

Para que el programa funcione tiene que estar compilado el fichero donde se encuentra la clase TCargador, y la clase que se origina tras la compilación debe estar en la ruta: "/home/condor/compartido/".

Se ejecuta java con la opción -cp (o -classpath), en la que se indica el directorio en el que están las clases (/home/condor/compartido/), pasándole como argumento la clase a ejecutar, en nuestro caso TCargador.

6.3.3.5.2. Windows

El código del programa es el siguiente

```
C:\jdk1.2.2\bin\java -cp c:\Condor\bin\ TCargador FINALIZAR
```

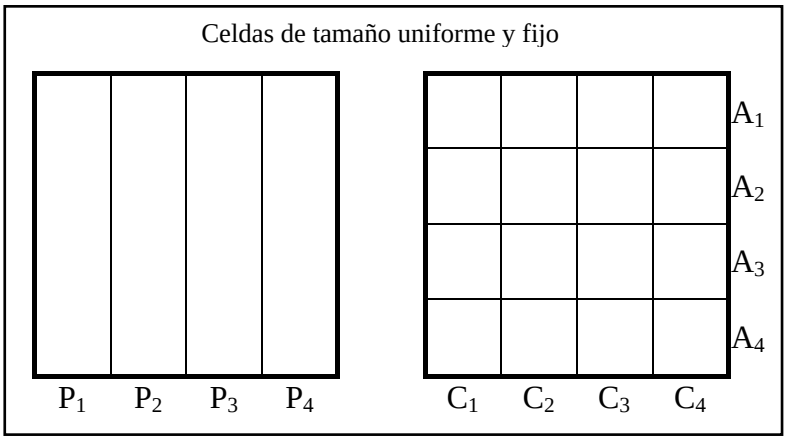
Este programa es un .bat y lo que hace es ejecutar java que se encuentra en la ruta: "C:\jdk1.2.2\bin\" con la opción –cp para indicarle que las clases están en el directorio "c:\Condor\bin\", pasándole como argumento la clase a ejecutar: TCargador, y el argumento de TCargador que es FINALIZAR.

6.4. MODIFICACIONES PARA OTRAS GESTIONES DE HARDWARE

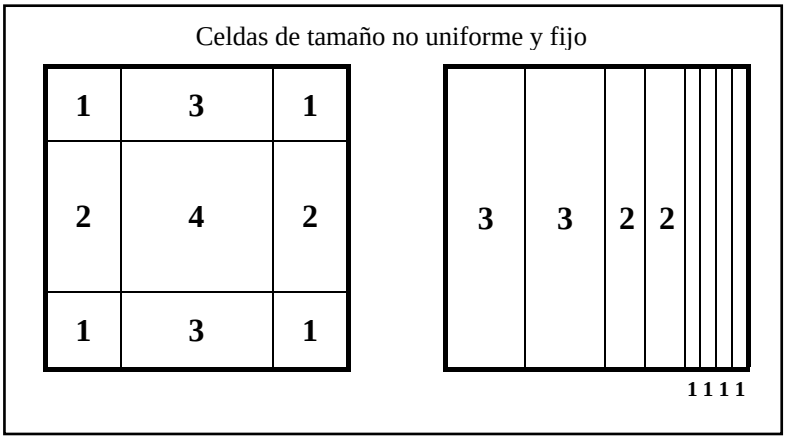
En el [Apartado 3] se habló de los distintos tipos de gestión de hardware dinámicamente reconfigurable, y también anteriormente, y como se ha podido comprobar en los anteriores puntos de este [Apartado 6], se dijo que el tipo de gestión que se iba a implementar era la más simple, la gestión estática. Pero como también se dijo, no sería demasiado complicado modificar ligeramente algunas cosas sobre el trabajo ya realizado y poder así utilizar Condor con FPGAs cuyos tipos de gestión son ya algo más complejos. Es a dichas modificaciones a las que dedicamos este apartado, y en ellas nos centraremos concretamente en su uso en los atributos de los ClassAds, ya que son la principal limitación, puesto que la forma de actualizar el estado de la FPGA (estado que se comunicará posteriormente a Condor), será tarea del gestor local de la FPGA y al poderse hacer en un lenguaje de alto nivel como puede ser Java será mucho más sencillo.

Comentaremos de pasada, por ser el que hemos visto, el tipo de gestión implementado. Este tipo consiste simplemente en indicar si la FPGA está en uso o no, para lo que rápidamente viene a la mente un booleano. Así pues, bastará con que el atributo FPGA_STATUS del ClassAd de la máquina que nos interesa tome el valor TRUE o FALSE, dependiendo si la FPGA está libre o no, respectivamente.

Pasemos ahora a la gestión dinámica, y empecemos por el modelo con celdas de tamaño uniforme y fijo. Puesto que todas las particiones tienen el mismo tamaño, no hace falta distinguirlas, y únicamente nos interesará saber si en la FPGA tiene alguna partición ocupada o libre, esto es, nos bastará con que el atributo FPGA_STATUS indique el número de particiones libres. Obviamente, podemos del mismo modo hacer que dicho atributo guarde el número de particiones ocupadas en vez de el número de particiones libres, pero es preferible hacerlo con el número de particiones libres dado que es más intuitivo poner en los requisitos del trabajo FPGA_STATUS >= 1, si las tareas siempre ocupan menos de una partición, o FPGA_STATUS >= X, donde X es el número de particiones que necesitamos (en el caso de querer mandara mas de una tarea que se han de ejecutar a la vez). Este tipo de implementación serviría tanto para el modelo de una como de dos dimensiones, o sea, cualquiera de estos dos modelos:



Si lo que nos interesa es utilizar el modelo con celdas de tamaño no uniforme y fijo, también es posible con sólo hacer una sencilla modificación. Para ello veamos un ejemplo; seguiremos el ejemplo con la figura del modelo de dos dimensiones pero este tipo de implementación sirve tanto para el modelo de una como para el de dos dimensiones.



Como vemos, tenemos la FPGA dividida en 9 particiones de 4 tipos distintos: cuatro del tipo 1, dos del tipo 2, dos del tipo 3, y una del tipo 4. Si nos fijamos bien, cuando tengamos un trabajo que requiera una partición del tipo 1, por ejemplo, nos estaremos poniendo de nuevo en el caso de las celdas de tamaño uniforme y fijo, puesto que como las cuatro celdas de ese tipo son iguales sólo me interesa saber cuántas hay libres. Y lo mismo en los otros tres casos. Así pues, lo más lógico sería llevar una lista, pero en los atributos de los ClassAds eso no puede ser, así que la idea es utilizar simplemente un número que indique cuántas particiones libres hay de cada tipo en el orden 1234, esto es, si la FPGA de la figura está completamente libre, la representación de su estado sería 4221. Con esta representación es sencillo escribir los requisitos, ya que si por ejemplo necesito una partición de tipo 3, me bastará con dividir el número entre 10 y ver después su módulo 10; si es mayor que 1 la FPGA será adecuada para el trabajo.

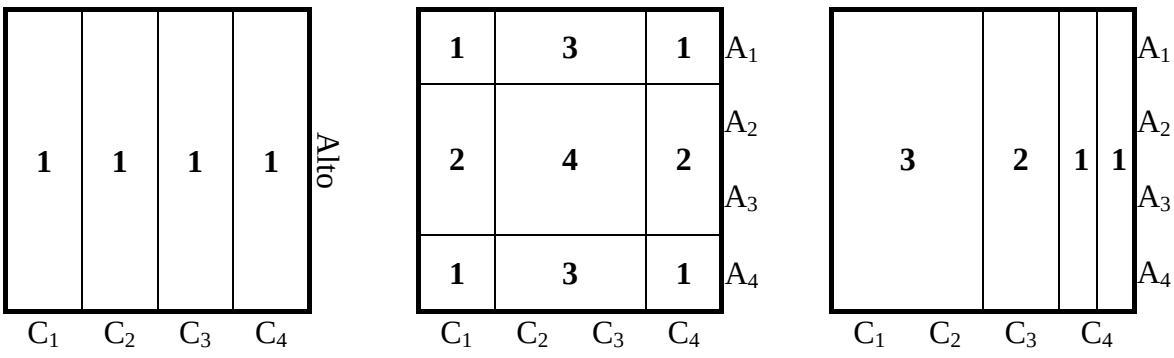
Si decidimos utilizar el modelo con celdas de tamaño variable de una dimensión podemos utilizar la misma estrategia que para el modelo con celdas de tamaño uniforme y fijo, ya que es un caso particular en el que las particiones, en lugar de tener un determinado ancho, tienen un ancho de un píxel. Obviamente a nivel del gestor local de la FPGA la implementación no es ni mucho menos tan sencilla, pero a nivel de Condor sólo nos interesa mantener la información de disponibilidad, así que bastaría, en lugar de indicar en FPGA_STATUS el número de particiones libres, indicar la anchura del mayor bloque libre, esto es, el mayor número de columnas adyacentes libres en la FPGA, y para poder dar una mayor flexibilidad, también podríamos añadir otros dos atributos, los cuales representen si la FPGA posee la posibilidad de desfragmentarse y el tamaño máximo que podría tener una partición en el caso de desfragmentar, con esto aumentamos la posibilidad del match con solo tener en cuanta la fragmentación que se genera en este tipo de gestión.

Por último, tenemos el modelo de celdas de tamaño variable de dos dimensiones. Si bien es cierto que podría seguirse la misma estrategia del caso anterior (FPGA_STATUS indica el mayor número de bloques libres adyacentes), el gestor local, para calcular este dato en función de sus listas de vértices y toda su información, tendría que realizar bastantes cálculos, y por tanto puede que ésta no sea la mejor opción. Otra opción pasaría por entrar ya a modificar el código de Condor para que pueda usar estructuras de datos más complejas y permitir de este modo un mejor traslado de información del gestor local de la FPGA al monitor de recursos de Condor puesto que bastaría con utilizar las mismas estructuras (o parecidas) con la misma información en ambas partes. En cualquier caso, como ya dijimos en el [Apartado 3] al hablar de este tipo de gestión, no se plantea como una opción aprovechable en estos momentos.

Por otro lado, como hemos dicho al comienzo del apartado, además de los distintos tipos de información que se guardaría en el atributo FPGA_STATUS, habría que realizar cambios en el gestor local de la FPGA para que pueda adaptar su información a su respectivo modo de representación en el atributo de los ClassAds de Condor, aunque no trataremos aquí dichos cambios puesto que, a pesar de que deberían de ser pequeños algoritmos ya en un lenguaje de alto nivel (o sea, sin tantas limitaciones como con los ClassAds), están fuera del alcance de nuestro proyecto. En cualquier caso, en nuestro proyecto, al pasar del modelo sencillo que se ha implementado a estos otros modelos de gestión más complejos, aparte de hacer los cambios pertinentes en cuanto a la información que guarda el atributo FPGA_STATUS, sería necesario añadir también una variable al simulador para guardar el estado actual de la supuesta FPGA (puesto que en el implementado, al variar siempre de TRUE a FALSE y viceversa esto no es necesario, basta con cambiar de un estado a otro directamente).

También queremos dejar constancia de la importancia de la geometría de las tareas, que aunque no se ha hablado de ella, seria fácilmente implementable en el ClassAd con solo introducir un atributos que representen el tipo de FPGA de que se dispone y su gestión, para así poder tenerlo en cuenta el los requisitos de las tareas y poder decir que una tarea se puede ejecutar en una FPGA de tipo X si tiene una partición Y libre etc…, lo cual es muy beneficioso, pues no es lo mismo una tarea cuadrada que alargada y no por tener una partición libre, implica que la tarea entre en ella pues se ha de especificar el tamaño de la tarea o mejor dicho restringir los Tipos de FPGAs e a solo aquellos en los que sabemos que nuestra tarea entra en alguna de las particiones que solicitamos.

Vamos a poner un ejemplo para entenderlo mas fácilmente: Queremos ejecutar una tarea, que tiene una geometría rectangular de alto 4 y ancho 2, figura 0, y disponemos de tres tipos de FPGAs, Figura 1,2 y 3, por lo cual en los requisitos de la tarea deberíamos de indicar, que si la FPGA es de Tipo 1 y tiene alguna partición libre o, si es de Tipo 2 y tiene alguna partición libre de tipo 2 o 4 o en el ultimo caso si es de Tipo 3 y tiene alguna partición libre de tipo 3 o 2.




```
Requirements =
(Type_FPGA==1 && FPGA_STATUS>=1)||
(Type_FPGA==2 && (FPGA_STATUS%10>=1)||((FPGA_STATUS/100)%10>=1))||
(Type_FPGA==3 && (FPGA_STATUS%10>=1)||((FPGA_STATUS/10)%10>=1))
```

7. POSIBLES MEJORAS

En este proyecto, el principal objetivo era conseguir la ejecución (o simulación) de tareas hardware a través de una red de equipos, con tarjetas FPGA y utilizando Condor, el cual se ha conseguido, pero a su vez hemos creído conveniente dejar presentes un conjunto de posibles mejoras que se podrían hacer en un futuro, las cuales son las siguientes:

7.1. OBTENCIÓN DE RESULTADOS

La propuesta de solución para la gestión de hardware dinámicamente reconfigurable que se ha implementado no soporta directamente la obtención de resultados para tareas hardware, a pesar de que el propio Condor permite directamente llevar un control de algunos sucesos que les ocurren a las tareas, tales como el momento de comienzo de la ejecución, la máquina en la que se ejecuta, las máquinas que la han rechazado y por qué, el momento de finalización de la ejecución y otros muchos…

Esto es debido a la estructura realizada y las limitaciones de Condor, pues la implementación desarrollada consiste primordialmente en el envío de una tarea consistente en un ejecutable, el cargador, cuyo propósito no es otro que enviar al gestor local de la FPGA los datos necesarios para que inserte una nueva tarea en su área de proceso. Estas tareas llegan al gestor local cuando es seguro que se van ha ejecutar en esa FPGA, es decir solamente se utiliza Condor como planificador, esto es, Condor es capaz de llevar un registro de esta tarea software pero no de la tarea hardware real, por lo cual sería necesario en un futuro, suponemos que el año próximo, la creación de algún mecanismo que pueda llevar este control real de las tareas hardware para luego poder devolver los resultados.

Este mecanismo podría ser, por ejemplo, la creación de un archivo a modo de log por parte del gestor local en algún sitio definido por la misma tarea o compartido por todos los usuarios; otra posible propuesta sería la ampliación de las variables pertenecientes al ClassAd de la máquina local para soportar la finalización o registro de sucesos de diversas tareas con alguna estructura de datos definida en formato String sobre una variable de Condor, de manera que Condor conozca el estado de dichas tareas para así poder ser éste aprovechado por los usuarios para poner restricciones de precedencia a las tareas u otros propósitos que se planteen.

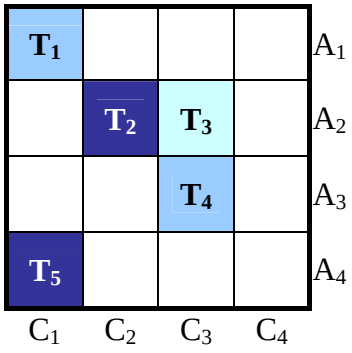
7.2. MONITORIZACIÓN DE LAS TAREAS

Como hemos comentado en el punto anterior, puede ser preciso ampliar las capacidades de Condor, para poder obtener datos acerca de las tareas en ejecución en las diversas FPGAs, esto sería fácilmente conseguible añadiendo en cada una de las máquinas que contengan una FPGA, una serie de variables al ClassAd, como podrían ser: Estado-Tareas, que representaría el estado de las tareas en ejecución, una posible estructura de datos sería un árbol o lista que contenga los datos necesarios, por ejemplo, el “tiempo de inicio” de la tarea, “tiempo de finalización”, “identificador”, “usuario” que la envió, “maquina” que la envió, porcentaje o área de FPGA utilizada o cualquier indicativo de “tamaño”, “partición” de la FPGA que la contiene,…

Un ejemplo sería:

```
FPGA_STATUS = 12
```

(La numeración de las particiones por columnas y filas (Ci, Ai), y el tamaño de cada partición es de 100 pixeles2)



```
JOB_STATUS = [ 5, Tarea (T0, Ghadir1, 147.96.81.67, 100, (4,4),
19/05/04 - 14:55, 19/05/04 - 15:23); Tarea (T1, Ghadir2,
147.96.81.68, 100, (1,1), 21/05/04 - 13:00, 0); Tarea (T2, Ghadir1,
147.96.81.67, 100, (2,2), 21/05/04 - 13:01, 0); Tarea (T3, Ghadir2,
147.96.81.68, 100, (2,3), 21/05/04 - 13:01, 0); Tarea (T4, Ghadir1,
147.96.81.67, 100, (3,3), 21/05/04 - 13:02, 0); Tarea (T5, Ghadir1,
147.96.81.67, 100, (1,4), 21/05/04 - 13:03, 0); ]
```

(La estructura de datos es. [#Tareas, Tarea;…, Tarea], siendo Tarea = Tarea (ID, usr, IP, tamaño, partición, inicio, fin), si el fin es 0, es porque aun esta activa)

Aparte de llevar esta estructura, necesitaríamos crear una aplicación o Script para poder consultarlos mas cómodamente, ya que el String podría ser muy grande y siempre se agradece que sea presentado con un formato adecuado o incluso sólo mostrando los datos que nos interesen, es decir más o menos le daríamos el uso al igual que a un Base de Datos, por lo cual también se podría estructurar como una cadena de texto en formato XML, que podríamos tratar a nuestro antojo.

7.3. PUNTOS DE CONTROL EN CONDOR (CONDOR'S CHECKPOINT MECHANISM)

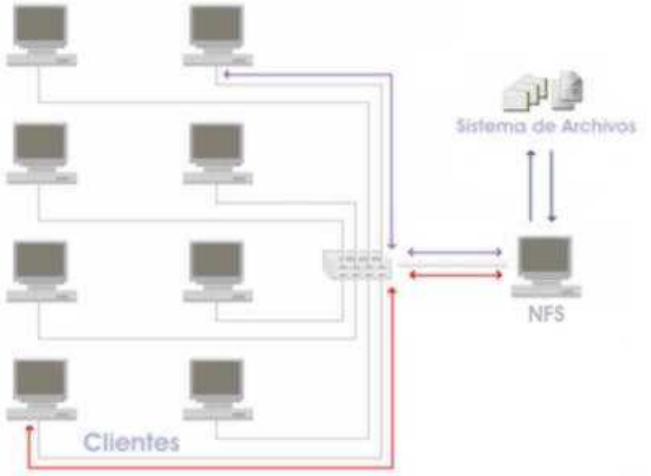
Esto es un mecanismo del que dispone Condor, aunque en nuestro caso no lo hemos configurado al no creerlo indispensable para nuestro propósito, sin embargo, de cara al futuro y a un uso real de esta aplicación, nos provee de características tan importantes como la mayor fiabilidad del sistema y el uso de expropiación en función de las prioridades de las tareas, lo cual es posible con la creación de puntos de control (Checkpoints).

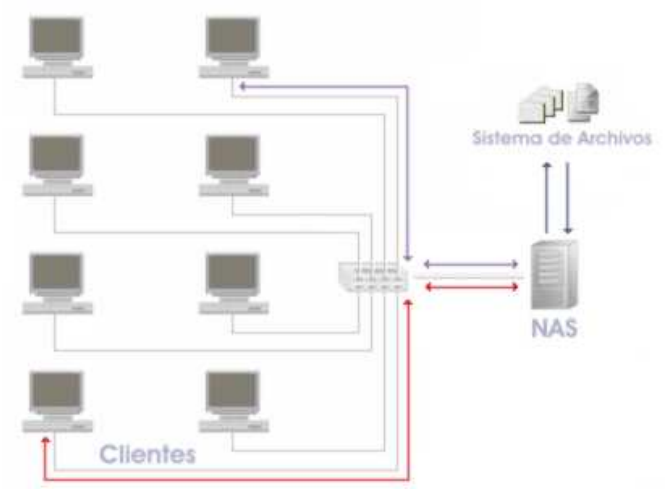
Estos puntos de control nos permitirían, desde parar una tarea en ejecución, por culpa de la entrada de una de mayor prioridad, a hacer que cada cierto tiempo se generen puntos de control de todo el sistema para así estar preparados para un posible fallo del sistema, sin embargo esta utilidad no nos vale directamente para las tareas hardware, ya que el control de estas lo tiene el gestor local, pero se podría modificar el mecanismo, para que Condor induzca a este gestor ha hacerlo mismo, es decir crear puntos de control, con lo cual nos permitiría aprovechar este mecanismo.

7.4. SISTEMA DE GESTIÓN DE ARCHIVOS POR LAN

Uno de los problema que plantea el diseño realizado, es la forma de transmitir los archivos necesarios desde el usuario que los quiere ejecutar hasta la maquina que los ejecuta definitivamente, este problema esta resuelto en nuestro caso utilizando un sistema de ficheros compartido, esta solución es buena siempre y cuando todas las maquinas de la red que deseen utilizar los recurso que tiene Condor a su disposición, dispongan del mismo sistema operativo, pues un Linux y un Windows, no pueden compartir un sistema de directorios común, a menos que se monte un servidor SAMBA, que nos permite compartir archivos entre distintos sistemas como Linux o Windows.

En estos momentos, están instalados, tanto un servidor NFS, para compartir archivos en Linux, como una red interna con archivos compartidos en Windows, esto permite el funcionamiento de nuestra aplicación junto con Condor, aunque no es posible mezclar sistemas con diversos S.O.





Otra opción, que seria la mas apropiada, aunque mas costosa, es un sistema de compartición de archivos por red, Network Attached Storage (NAS), el cual es como un disco duro conectado directamente a la red ya sea LAN o WAN, es decir nos permitiría en función de los permisos de usuario un acceso dentro de la misma red o incluso a través de Internet.

De esta manera tendríamos centralizados todos los datos y registros de Condor y de las tareas que este realiza, lo cual es muy beneficioso y simplifica en gran medida la utilización, y el mantenimiento del sistema.

7.5. TRANSMISIÓN DE ARCHIVOS A TRAVÉS DE CONDOR

Otra opción, puede ser utilizar Condor para transmitir directamente los ficheros necesarios, esta opción, no se ha investigado pues en su momento no fue necesario, pero seria interesante como alternativa al almacenamiento de red.

8. CONCLUSIONES

Tras haber trabajado durante este tiempo con Condor, hemos llegado a la conclusión de que si bien es un sistema que ofrece múltiples posibilidades, es posible que realmente no sea la solución más apropiada a la hora de gestionar unos determinados recursos hardware, a pesar de que se ha comprobado que es posible utilizarlo para tal fin. El que sea posible que sea o no la solución más apropiada depende de dónde se vaya a utilizar el sistema Condor con las modificaciones necesarias.

Si se pretende utilizar en una red grande y con vistas a, aparte de gestionar recursos hardware reconfigurables, gestionar también recursos software y poder realizar tareas que requieren grandes cantidades de cálculo distribuyéndolas, el uso de este sistema modificado sí sería una buena elección, ya que se está utilizando todo el potencial que condor ofrece, además de, claro está, la nueva funcionalidad de gestión de recursos hardware.

Si por el contrario lo único que nos interesa es gestionar los recursos hardware como pueden ser en este caso las FPGAs, Condor no es la solución más apropiada. Está claro que va a funcionar, ya se ha visto, pero estaremos desaprovechando multitud de opciones de Condor que ocupan espacio y complican su funcionamiento. Si bien el asunto del espacio en disco podría ser pasable dadas las actuales capacidades de almacenamiento de los discos duros, lo relacionado con la forma en que se complica el sistema no lo es tanto. Esto es, puede que falle algo que esté relacionado con una funcionalidad que no estamos usando, o puede que el fallo no esté ahí pero como no se puede descartar nada pues se perdería tiempo en descartar todo lo relacionado con dicha funcionalidad no utilizada a la hora de detectar el problema. Además de, claro está, complicar posibles extensiones puesto que hay que estar pendiente de que los cambios encajen con todo, y si ese “todo” es grande y complejo el trabajo será mucho mayor. En resumen, si sólo quisiésemos usar Condor con este fin, sería como utilizar un autobús para llevar a un solo niño al colegio: funcionaría, pero se le podría llevar en un coche normal.

Por otro lado, con Condor encontramos un par de dificultades, no relacionadas con el uso que demos al sistema y ya comentadas a lo largo de este documento, que quizás con un sistema diseñado “a medida” no encontraríamos.

Una es la imposibilidad de transmitir los ficheros de una a otra máquina, al menos de momento, puesto que, en teoría, Condor debería hacerlo, así que es posible que en futuras versiones (hemos trabajado con la 6.4.7) funcione correctamente en cualquier caso. De todos modos ya se trató este problema en el apartado de posibles mejoras ([Apartado 7]) y siempre se podría solucionar con alguna de las maneras ahí propuestas.

La otra dificultad con la que nos tropezamos es lo comentado en el [Apartado 6.4] acerca de los modelos de gestión con celdas de tamaño variable: si bien es cierto que cabría la posibilidad de representar en el ClassAd el estado de una FPGA con gestión de ese tipo, complica las cosas a nivel del gestor local para calcular lo que ha de enviar al ClassAd para representar la configuración de la FPGA en Condor, a no ser que, como dijimos, se opte por modificar el código de Condor.

En cualquier caso, consideramos que siempre es interesante realizar este tipo de estudios ya que aunque es posible, como en este caso, que el sistema no se adapte exactamente lo que queremos, se recaba información suficiente para otras futuras modificaciones parecidas que podrían querer hacerse y para poder modificar Condor rápidamente en caso de necesitar, a corto plazo, un planificador y administrador de recursos para un recurso determinado.

9. COMO ENLAZAR EL PROYECTO DENTRO DE GHADIR

Como se dijo en el [Apartado 1], este proyecto está englobado en un proyecto mayor, GHADIR, sobre la gestión de hardware dinámicamente reconfigurable. Asimismo, a lo largo de distintos apartados se ha hablado del gestor local de la FPGA. Dado que Condor es informado del estado de la FPGA por el gestor local (mediante el monitor de recursos externo del que se habló en el [Apartado 5.4]) y que por tanto están relacionados uno y otro, explicaremos aquí cómo enlazar este proyecto dentro del GHADIR, o lo que es lo mismo, como enlazar este proyecto con otra parte del GHADIR encargada de la gestión local.

Para enlazar este proyecto dentro del proyecto global, se nos han ocurrido dos opciones. Una de ellas es más sencilla y se limita a utilizar el gestor de la FPGA para ejecutar el trabajo como si fuese una aplicación cualquiera; la otra opción es algo más compleja ya que modifica un poco el gestor pero a cambio queda todo más unido y permite tipos de gestión más complejos.

La primera opción consiste en dejar el simulador desarrollado en este proyecto tal y como está, con la diferencia de que cuando llegue la tarea lo que hará será ejecutar, en lugar de los procesadores de texto que se usaron para las pruebas, el gestor local de la FPGA, al que se le pasarán como parámetros tanto el archivo de datos como el archivo de configuración necesarios. Obviamente para poder hacer esto necesitaríamos que el gestor de la FPGA pudiese ejecutarse de esta forma, esto es, tal y como un programa cualquiera al que se le llama con unos parámetros. Así que si el gestor necesitase estar ejecutándose continuamente esta opción no sería factible (ya que estamos pensando en el gestor como en un programa que se ejecuta cuando se le llama y luego acaba una vez acabado el trabajo), a menos que se implementase una función en el gestor local que se ocupase de este tipo de llamadas (o sea, cuando se hace una llamada de este tipo, se llama a esa función que es ya la que carga el trabajo). Pero claro, esto conllevaría modificar el gestor local, lo que haría perder toda la ventaja de esta primera opción frente a la segunda (esto es, no tocar el código del gestor).

Además, como hemos dicho, esta solución es sencilla pero sólo sirve para modelos de gestión sencillos, tales como la gestión estática o la gestión dinámica con celdas de tamaño uniforme y fijo, en el supuesto de que los trabajos siempre fuesen, a lo sumo, de una partición entera. Esto es debido a que el simulador ejecuta el gestor, pero una vez llamado se olvida de él y también tiene que llamar luego al Script o programa que hace las veces de monitor de recursos externo para actualizar el atributo FPGA_STATUS, y por tanto estos modelos son los únicos que nos permiten saber de antemano el valor que tomará (el valor contrario en caso de usar los booleanos, o el valor menos uno en el caso de las celdas de tamaño uniforme y fijo). Si quisiésemos utilizar esta solución con los otros modelos dinámicos, sería necesario utilizar alguna forma de comunicación entre el simulador y el gestor (probablemente sockets) y por tanto habría que modificar el gestor, igual que en la segunda opción, y por ello nuevamente esa sería preferible en ese caso.

La segunda solución consiste en integrar el hilo que es realmente el simulador en el gestor local de la FPGA, o sea, que hay que modificar un poco el gestor. La idea en este caso sería integrar el hilo en la aplicación del gestor de modo que el hilo se estuviese ejecutando por un lado y el resto de la aplicación por el otro, pero teniendo todos los datos referentes a la ocupación y el estado de la FPGA como globales, de forma que ya no serían necesarios los sockets ni ninguna de esas formas de comunicación. Si acaso, posiblemente fuese necesario algún mecanismo para proteger los accesos a dichos datos, pero nada más. En cuanto al modo de cargar los datos, se podrían sustituir las llamadas que había a los editores de texto con las que hicimos las pruebas (y que en la opción primera se sustituían simplemente por llamadas al gestor, esto es, seguían siendo llamadas a una aplicación) por alguna función interna, de modo que cuando el hilo "del simulador" llamase a esta función con los archivos necesarios para ejecutar el trabajo (por ejemplo, algo del tipo *cargar(bitmap,config)*), el trabajo se cargase en la FPGA cuyo control real posee el otro hilo, modificando los datos globales acerca del estado de la FPGA y pudiendo así actualizar correctamente el ClassAd en Condor. Como puede verse, esta solución da más trabajo pero proporciona una mayor funcionalidad y queda todo más integrado.

10. PRUEBAS REALIZADAS

10.1 PRIMER INTENTO DE PUESTA EN FUNCIONAMIENTO DEL POOL

Iniciamos el central manager con la orden condor_master, que se encarga de inicializar los demonios necesarios para el funcionamiento del pool (entre ellos el demonio condor_master) en la máquina ghadir2.

Comprobamos que se han iniciado correctamente los demonios con la orden:

```
ps -ef |egrep condor_
```

Deberán aparecer los siguientes demonios:

```
Condor      2089  1     0     17:58 ?     00:00:00    condor_master
condor      2090  2089  0     17:58 ?     00:00:00    condor_collector-f
condor      2091  2089  0     17:58 ?     00:00:00    condor_negotiator-f
condor      2092  2089  11    17:58 ?     00:00:01    condor_startd-f
condor      2093  2089  0     17:58 ?     00:00:00    condor_schedd-f
```

Iniciamos un cliente en la otra máquina (ghadir1)con la orden: condor_master

Comprobamos en el central manager (ghadir2) los componentes del pool con el comando condor_status:

```
Name  OpSys Arch  State Activity    LoadAv      Mem   ActvtyTime
ghadir2.dacya     LINUX INTEL Unclaimed   Idle  0.000 501   0+00:05:04

      Machines    Owner Claimed     Unclaimed   Matched     Preempting

INTEL/LINUX 1     0     0           1           0           0

      Total 1     0     0           1           0           0
```

Observamos que ghadir1 no aparece en el pool. Llegamos a la conclusión tras revisar la instalación de que ambas máquinas no pertenecen al mismo dominio, y que por lo tanto, no se ven. El fallo cometido en la instalación se produjo en el paso 5 donde creamos un dominio para cada máquina.

Para solucionarlo modificamos manualmente el fichero de configuración condor_config, sustituimos la siguiente línea:

```
UID_DOMAIN  = $(FULL_HOSTNAME)
```

Por esta otra:

```
UID_DOMAIN  = dacya.ucm.es
```

Tras este cambio en todas las máquinas, volvemos a realizar todo el proceso, y observamos que ahora al ejecutar la orden condor_status, aparecen ambas máquinas en el pool:

```
Name              OpSys   Arch    State   Activity LoadAv Mem     ActvtyTime

ghadir1.dacya     WINNT51 INTEL   Unclaimed Idle  0.052   512     0+00:43:03
ghadir2.dacya     WINNT51 INTEL   Unclaimed Idle  0.031   512     0+00:20:04

                  Machines Owner  Claimed Unclaimed Matched Preempting

INTEL/WINNT51     2         0     0       2         0       0

          Total   2         0     0       2         0       0
```

10.2 EJECUCIÓN DE TAREAS EN UNA SOLA MÁQUINA

El objetivo de este apartado es familiarizarnos con Condor y los ficheros de submit con la Ejecución de tareas en una sola máquina

10.2.1 TAREAS ESCRITAS EN C

10.2.1.1 Primera prueba

1. Realizamos un programa sencillo p1.c:

```
#include <stdio.h>
int main()
{
      FILE* f;
      f=fopen("p1.txt","w+");
      fprintf(f,"Esto es una prueba\n");
      fclose(f);
      return 1;
}
```

Este programa lo único que hace es crear un fichero p1.txt con la siguiente información: Esto es una prueba.

2. Compilamos el programa con gcc, con la siguiente orden: gcc -o p1 p1.c
3. Ejecutamos el programa y comprobamos q funciona correctamente.
4. Ahora el siguiente paso consiste en ejecutar la tarea utilizando Condor, para ello lo primero que debemos hacer es generar un fichero asociado al envío llamado “submit.description file” (fichero de descripción de envió), en el cual se describen los parámetros y requisitos necesarios para la ejecución de la tarea en el pool, en nuestro caso es el siguiente: submit.p1

```
#####################################
#                                   #
# Prueba 1                          #
# Fichero de Descripción del Job    #
#                                   #
#####################################

Executable  = p1
Universe    = vanilla
Log         = result/log/p1.log
Queue
```

Simplemente, indicamos en el argumento Executable, el nombre del ejecutable: p1, en Universe, indicamos que utilizamos el universo vanilla, e indicamos que los ficheros de log lo debe guardar en result/log/p1.log. Queue es para que introduzca la tarea en la cola.

5. Enviamos la tarea con la orden condor_submit submit.p1
   El resultado obtenido es el siguiente:

```
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 54.
```

6. Comprobamos que la tarea está en la cola con la orden condor_q:

```
-- Submitter:ghadir2.dacya.ucm.es:<147.96.81.68:33884>:ghadir2.dacya.ucm.es
ID    OWNER       SUBMITTED   RUN_TIME          ST    PRI   SIZE  CMD
54.0  condor      2/5         18:42 0+00:00:00  I     0     0.0   p1

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

7. Observamos que la tarea está en estado idle y que no se ejecuta. Para ver cual es la causa de que no se ejecute, ejecutamos la orden condor_q -analyze, la cual nos informa del motivo:

```
-- Submitter:ghadir2.dacya.ucm.es:<147.96.81.68:33884>:ghadir2.dacya.ucm.es
ID    OWNER       SUBMITTED   RUN_TIME          ST    PRI   SIZE  CMD
---
054.000:  Run analysis summary.  Of 1 machines,
      0 are rejected by your job's requirements
      1 reject your job because of their own requirements
      0 match, but are serving users with a better priority in the
pool
      0 match, but prefer another specific job despite its worse
user-priority
      0 match, but cannot currently preempt their existing job
      0 are available to run your job
        No successful match recorded.
        Last failed match: Thu Feb  5 18:42:43 2004
        Reason for last match failure: no match found

WARNING:  Be advised:   Request 54.0 did not match any resource's
constraints


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

El motivo es que no se puede ejecutar por los requisitos de nuestra tarea, tras consultar el fichero condor_config nos damos cuenta de que el problema está en la variable START. En el condor_config aparece:

```
START       = $(UWCS_START)
UWCS_START  = ( (KeyboardIdle > $(StartIdleTime)) \
                    && ( $(CPUIdle) || \
                         (State != "Unclaimed" && State != "Owner"))
)
StartIdleTime     = 15 * $(MINUTE)
MINUTE            = 60
CPUIdle           = ($(NonCondorLoadAvg) <= $(BackgroundLoad))
NonCondorLoadAvg  = (LoadAvg - CondorLoadAvg)
BackgroundLoad    = 0.3
```

Lo cual indica que las tareas deben ejecutarse cuando hayan pasado 15minutos de inactividad del teclado o cuando la cpu no este siendo utilizada. Para especificar que las tareas pasen a ejecutarse inmediatamente, sustituimos la línea anterior por la siguiente línea:

```
START = TRUE
```

10.2.1.2 Segunda prueba

1. Realizamos el siguiente programa en c:

```
#include <stdio.h>

int main(void)
{
    printf("hello, Condor\n");
    return 0;
}
```

Este programa lo único que hace es escribir por pantalla, el objetivo es probar el funcionamiento de la redirección de la salida (stdout), es decir comprobar que la salida, que debería darse por pantalla, Condor se encarga de escribirla en un fichero.

2. Compilamos el programa con gcc, con la siguiente orden: gcc -o p2 p2.c

3. Ejecutamos el programa y comprobamos q funciona correctamente.

4. Creamos el fichero submit.p2:

```
######################################
#                                    #
# Prueba 2                           #
# Fichero de Descripción del Job     #
#                                    #
######################################

Executable  = p2
Universe    = vanilla
Output      = result/out/p2.out
Log   = result/log/p2.log
Queue
```

La novedad es que hemos añadido el parámetro Output, con el cual le indicamos a Condor donde debe de depositar la salida estándar de la tarea, en caso de omisión de este parámetro dicha salida se redirecciona a /dev/null.

5. Enviamos la tarea con la orden condor_submit submit.p2
   El resultado obtenido es el siguiente:

```
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 55.
```

6. Comprobamos que la tarea está en la cola con la orden condor_q:

```
-- Submitter:ghadir2.dacya.ucm.es:<147.96.81.68:34301>:ghadir2.dacya.ucm.es
ID    OWNER       SUBMITTED   RUN_TIME          ST    PRI   SIZE  CMD
55.0  condor      2/5         19:15 0+00:00:00  I     0     0.0   p2

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

Observamos que la tarea está en estado idle y pero al poco tiempo se ejecuta, además tras ejecutar la orden condor_q -analyze, se observa que la tarea no ha sido rechazada, como antes:

```
-- Submitter:ghadir2.dacya.ucm.es:<147.96.81.68:34301>:ghadir2.dacya.ucm.es
ID    OWNER       SUBMITTED   RUN_TIME          ST    PRI   SIZE  CMD
---
055.000:  Run analysis summary.  Of 1 machines,
      0 are rejected by your job's requirements
      0 reject your job because of their own requirements
      0 match, but are serving users with a better priority in the
pool
      1 match, but prefer another specific job despite its worse
user-priority
      0 match, but cannot currently preempt their existing job
      0 are available to run your job
        Last successful match: Thu Feb  5 19:16:11 2004

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

7. Comprobamos que la tarea se ha ejecutado y generado los archivos adecuados, con la orden condor_q vemos q ya no hay ninguna tarea esperando a ejecutarse, y que en /home/condor/result/out/ esta el fichero p2.out, que contiene la salida estándar de nuestro programa, como esperábamos, también podemos observar el fichero de log el cual nos informa de que la tarea se ha ejecutado. El contenido del fichero de log es el siguiente:

```
000 (055.000.000) 02/05 19:13:42 Job submitted from host:
<147.96.81.68:34301>
...
001 (055.000.000) 02/05 19:14:00 Job executing on host:
<147.96.81.68:34300>
...
005 (055.000.000) 02/05 19:14:00 Job terminated.
      (1) Normal termination (return value 0)
            Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
      0  -  Run Bytes Sent By Job
      0  -  Run Bytes Received By Job
      0  -  Total Bytes Sent By Job
      0  -  Total Bytes Received By Job
...
```

10.2.2 TAREAS ESCRITAS EN JAVA

10.2.2.1 Primera prueba

1. Realizamos un programa sencillo hello.java:

```
public class hello {
    public static void main( String[] args )
    {
        System.out.println( "Hello World" );
    }
}
```

Este programa lo único que hace es mostrar por pantalla el mensaje “Hello Word”.

2. Compilamos el programa con javac, con la siguiente orden: javac hello.java, que genera hello.class, el cual es necesario ejecutarlo con java.

3. Ejecutamos el programa y comprobamos que funciona correctamente con la orden java hello.

4. Generamos el fichero submit.hello

```
####################################
#                                  #
# Prueba 1 en Java                 #
# Fichero de Descripción del Job   #
#                                  #
####################################

Executable  = hello.class
Universe    = java
Output           = result/out/hello.out
Log         = result/log/hello.log
Queue
```

La única diferencia con las tareas en C es el universo que ha de ser Java.

5. Enviamos la tarea con la orden condor_submit submit.hello. El resultado obtenido es el siguiente:

```
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 59.
```

6. Comprobamos que la tarea está en la cola con la orden condor_q:

```
-- Submitter: ghadir2.dacya.ucm.es : <147.96.81.68:34301> :
ghadir2.dacya.ucm.es ID      OWNER            SUBMITTED     RUN_TIME
ST PRI SIZE CMD
  59.0   condor          2/5  19:37   0+00:00:02 I  0   0.0  java

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

Tras un tiempo de espera y al ver que la tarea no se ejecuta procedemos a ejecutar condor_q -analyze:

```
--
Submitter:ghadir2.dacya.ucm.es:<147.96.81.68:34301>:ghadir2.dacya.ucm.es
ID    OWNER       SUBMITTED   RUN_TIME          ST    PRI   SIZE  CMD
---
059.000:  Run analysis summary.  Of 1 machines,
      0 are rejected by your job's requirements
      0 reject your job because of their own requirements
      0 match, but are serving users with a better priority in the
pool
      1 match, but prefer another specific job despite its worse
user-priority
      0 match, but cannot currently preempt their existing job
      0 are available to run your job
        Last successful match: Thu Feb  5 19:37:35 2004

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

La tarea no es rechazada, pero no pasa a ejecución, para intentar que pase a ejecución, probamos a darle mayor prioridad, pero esto tampoco soluciona el problema, investigando la estructura de los ficheros de submit, nos dimos cuenta de que en tareas java es necesario poner el parámetro arguments = hello, que es el argumento que se le pasa a la maquina virtual de java. Así pues el fichero de submit queda de la siguiente forma:

```
#####################################
#                                   #
# Prueba 1 en Java                  #
# Fichero de Descripción del Job    #
#                                   #
#####################################

Executable  = hello.class
Universe    = java
Arguments   = hello
Output            = result/out/hello.out
Log         = result/log/hello.log
Queue
```

7. Comprobamos que la tarea se ha ejecutado y generado los archivos adecuados, con la orden condor_q vemos q ya no hay ninguna tarea esperando a ejecutarse, y que en /home/condor/result/out/ esta el fichero hello.out, que contiene la salida estándar de nuestro programa, también podemos observar el fichero de log el cual nos informa de que la tarea se ha ejecutado:

```
000 (059.000.000) 02/05 19:45:34 Job submitted from host:
<147.96.81.68:34301>
...
001 (059.000.000) 02/05 19:45:37 Job executing on host:
<147.96.81.68:34300>
...
005 (059.000.000) 02/05 19:45:37 Job terminated.
      (1) Normal termination (return value 0)
            Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
      0  -  Run Bytes Sent By Job
      0  -  Run Bytes Received By Job
      0  -  Total Bytes Sent By Job
      0  -  Total Bytes Received By Job
```

10.2.2.2 Segunda prueba

1. Realizamos un programa sencillo helloFile.java:

```
import java.io.*;
public class helloFile {
    public static void main( String[] args )
    {
       String filename = "result/out/helloFile.txt";
          try
          { // opens try
            BufferedWriter bw = new BufferedWriter
             (new FileWriter(filename));
            bw.write("HelloFile");
            bw.flush();
            bw.close();
           } // closes try
         catch (IOException ioe)
          { // open catch
                    System.out.println("Error");
          }// close catch
     }
}
```

   Este programa lo único que hace es escribir un fichero con el texto “HelloFile”.

2. Compilamos el programa con el comando javac de la siguiente forma: javac helloFile.java. Esto genera un fichero helloFile.class, el cual es necesario ejecutarlo con java.

3. Ejecutamos el programa y comprobamos q funciona correctamente con el comando: java helloFile.

4. Generamos el fichero submit.helloFile

```
####################################
#                                  #
# Prueba 2 en Java                 #
# Fichero de Descripción del Job   #
#                                  #
####################################

Executable  = helloFile.class
Priority    = 20
Arguments   = helloFile
Universe    = java
Output           = result/out/helloFile.out
Log         = result/log/helloFile.log
Queue
```

5. Enviamos la tarea con la orden condor_submit submit.helloFile. El resultado es el siguiente:

```
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 65.
```

6. Comprobamos que la tarea está en la cola con la orden condor_q:

```
-- Submitter:ghadir2.dacya.ucm.es:<147.96.81.68:35260>:ghadir2.dacya.ucm.es
ID    OWNER       SUBMITTED   RUN_TIME          ST    PRI   SIZE  CMD
65.0  condor      2/5         20:10 0+00:00:02  R     20    0.0   java helloFile

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

7. Comprobamos que la tarea se ha ejecutado y generado los archivos adecuados. Con la orden condor_q vemos que ya no hay ninguna tarea esperando a ejecutarse, que en /home/condor/result/out/ esta el fichero helloFile.txt, y que además, al redirigir la salida estándar también hay un fichero helloFile.out, aunque en este caso esta vació ya que no hay salida estándar, también podemos observar el fichero de log el cual nos informa de que la tarea se ha ejecutado:

```
000   (065.000.000)   02/05   20:10:42   Job   submitted   from   host:
<147.96.81.68:35260>
...
001   (065.000.000)   02/05   20:11:02   Job   executing   on   host:
<147.96.81.68:35259>
...
005 (065.000.000) 02/05 20:11:03 Job terminated.
      (1) Normal termination (return value 0)
            Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
      0  -  Run Bytes Sent By Job
      0  -  Run Bytes Received By Job
      0  -  Total Bytes Sent By Job
      0  -  Total Bytes Received By Job
```

10.3 EJECUCIÓN DE TAREAS EN VARIAS MÁQUINAS

El objetivo de este apartado es familiarizarnos con condor y la Ejecución de tareas en varias máquinas

Lanzamos varias tareas desde ambas maquinas y como esperábamos las tareas se reparten entre ambas maquinas, esto lo podemos observar en los ficheros de log creados, en donde se informa de en que maquina se ha ejecutado cada tarea:

```
000 (012.000.000) 02/09 17:57:55 Job submitted from host:
<147.96.81.67:34211>
...
000 (012.001.000) 02/09 17:57:55 Job submitted from host:
<147.96.81.67:34211>
...
001 (012.000.000) 02/09 17:57:58 Job executing on host:
<147.96.81.67:34210>
...
005 (012.000.000) 02/09 17:57:58 Job terminated.
      (1) Normal termination (return value 0)
            Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
      0  -  Run Bytes Sent By Job
      0  -  Run Bytes Received By Job
```

```
      0  -  Total Bytes Sent By Job
      0  -  Total Bytes Received By Job
...
001 (012.001.000) 02/09 17:58:00 Job executing on host:
<147.96.81.68:33605>
...
006 (012.001.000) 02/09 17:58:08 Image size of job updated: 73852
...
005 (012.001.000) 02/09 17:58:12 Job terminated.
      (1) Normal termination (return value 0)
            Usr 0 00:00:07, Sys 0 00:00:00  -  Run Remote Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
            Usr 0 00:00:07, Sys 0 00:00:00  -  Total Remote Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
      0  -  Run Bytes Sent By Job
      0  -  Run Bytes Received By Job
      0  -  Total Bytes Sent By Job
      0  -  Total Bytes Received By Job
...
```

Como vemos en el fichero de log anterior hay algunos trabajos que se han ejecutado en 147.96.81.67 y otros en 147.96.81.68.

10.4 PRUEBA DEL FUNCIONAMIENTO DE LOS REQUIREMENTS

Se realiza el siguiente fichero de submit para comprobar el funcionamiento de los requeriments:

```
####################################
#                                  #
# Prueba 1 en Java                 #
# Fichero de Descripcion del Job   #
#                                  #
####################################

Executable  = hello.class
Requirements      = HasFPGA==TRUE
Priority    = 20
Arguments   = hello
Universe    = java
Output            = result/out/hello.out
Log         = result/log/hello.log
transfer_files    = ONEXIT
Queue
```

Se ejecuta el mismo programa que el visto en el [Apartado 10.2.2.1], pero ahora se restringen las máquinas en las que se puede ejecutar, ahora sólo se puede ejecutar en las máquinas que tengan la variable HasFPGA a cierto.

Se comprueba que el resultado es el correcto poniendo en una máquina esa variable a false y viendo que por más veces que enviemos esa tarea, siempre la ejecuta en la máquina que tiene esa variable a true.

Por ejemplo, si ponemos HasFPGA = false en la máquina 147.96.81.68 y enviamos un trabajo dos veces, las dos veces lo ejecutará en la máquina 147.96.81.67:

```
000 (109.000.000) 05/07 16:31:45 Job submitted from host:
<147.96.81.68:32797>
...
000 (110.000.000) 05/07 16:36:05 Job submitted from host:
<147.96.81.68:32797>
...
001 (109.000.000) 05/07 16:36:10 Job executing on host:
<147.96.81.67:32785>
...
005 (109.000.000) 05/07 16:36:10 Job terminated.
      (1) Normal termination (return value 0)
            Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
      12  -  Run Bytes Sent By Job
      415  -  Run Bytes Received By Job
      12  -  Total Bytes Sent By Job
      415  -  Total Bytes Received By Job
...
001 (110.000.000) 05/07 16:36:12 Job executing on host:
<147.96.81.67:32785>
...
005 (110.000.000) 05/07 16:36:13 Job terminated.
      (1) Normal termination (return value 0)
            Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
            Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
      12  -  Run Bytes Sent By Job
      415  -  Run Bytes Received By Job
      12  -  Total Bytes Sent By Job
      415  -  Total Bytes Received By Job
...
```

10.5 PRUEBA FINAL DEL FUNCIONAMIENTO DE TODO

El estado de las máquinas para hacer la prueba es el siguiente:

```
C:\Documents and Settings\GHADIR\Mis documentos\Submits>condor_status
-l
MyType = "Machine"
TargetType = "Job"
Name = "ghadir1.dacya.ucm.es"
Machine = "ghadir1.dacya.ucm.es"
Rank = 0.000000
CpuBusy = ((LoadAvg - CondorLoadAvg) >= 0.500000)
FPGA_STATUS = TRUE
HasFPGA = TRUE
CondorVersion = "$CondorVersion: 6.4.7 Jan 27 2003 $"
CondorPlatform = "$CondorPlatform: INTEL-WINNT40 $"
VirtualMachineID = 1
VirtualMemory = 545388
Disk = 99936
CondorLoadAvg = 0.000655
LoadAvg = 0.632079
KeyboardIdle = 138
ConsoleIdle = 138
```

```
Memory = 512
Cpus = 1
StartdIpAddr = "<147.96.81.67:1029>"
Arch = "INTEL"
OpSys = "WINNT51"
UidDomain = "dacya.ucm.es"
FileSystemDomain = "dacya.ucm.es"
Subnet = "147.96.81"
HasIOProxy = TRUE
TotalVirtualMemory = 545388
TotalDisk = 99936
KFlops = 190212
Mips = 800
LastBenchmark = 1088431242
TotalLoadAvg = 0.632079
TotalCondorLoadAvg = 0.000655
ClockMin = 1016
ClockDay = 1
TotalVirtualMachines = 1
HasFileTransfer = TRUE
HasMPI = TRUE
JavaVendor = "Sun Microsystems Inc."
JavaVersion = "1.2.2"
JavaMFlops = 87.084091
HasJava = TRUE
StarterAbilityList = "HasFileTransfer,HasMPI,HasJava"
CpuBusyTime = 62
CpuIsBusy = TRUE
State = "Unclaimed"
EnteredCurrentState = 1088434530
Activity = "Idle"
EnteredCurrentActivity = 1088434530
Start = TRUE
Requirements = START
CurrentRank = 0.000000
UpdateSequenceNumber = 18
DaemonStartTime = 1088431228
LastHeardFrom = 1088434534

MyType = "Machine"
TargetType = "Job"
Name = "ghadir2.dacya.ucm.es"
Machine = "ghadir2.dacya.ucm.es"
Rank = 0.000000
CpuBusy = ((LoadAvg - CondorLoadAvg) >= 0.500000)
FPGA_STATUS = TRUE
HasFPGA = FALSE
CondorVersion = "$CondorVersion: 6.4.7 Jan 27 2003 $"
CondorPlatform = "$CondorPlatform: INTEL-WINNT40 $"
VirtualMachineID = 1
VirtualMemory = 513708
Disk = 665900
CondorLoadAvg = 0.000000
LoadAvg = 0.040487
KeyboardIdle = 0
ConsoleIdle = 0
Memory = 512
Cpus = 1
StartdIpAddr = "<147.96.81.68:1030>"
```

```
Arch = "INTEL"
OpSys = "WINNT51"
UidDomain = "dacya.ucm.es"
FileSystemDomain = "dacya.ucm.es"
Subnet = "147.96.81"
HasIOProxy = TRUE
TotalVirtualMemory = 513708
TotalDisk = 665900
KFlops = 155707
Mips = 646
LastBenchmark = 1088428862
TotalLoadAvg = 0.040487
TotalCondorLoadAvg = 0.000000
ClockMin = 1011
ClockDay = 1
TotalVirtualMachines = 1
HasFileTransfer = TRUE
HasMPI = TRUE
StarterAbilityList = "HasFileTransfer,HasMPI"
CpuBusyTime = 0
CpuIsBusy = FALSE
State = "Unclaimed"
EnteredCurrentState = 1088428862
Activity = "Idle"
EnteredCurrentActivity = 1088428862
Start = TRUE
Requirements = START
CurrentRank = 0.000000
UpdateSequenceNumber = 18
DaemonStartTime = 1088428836
LastHeardFrom = 1088434266
```

Se han subrayado las variables importantes.

Se realiza el siguiente fichero de submit:

```
#####################################
#                                   #
# Submit para el Cargador           #
# Fichero de Descripción del Job    #
#                                   #
#####################################

Executable  = TCargador.class
Requirements      = HasFPGA==TRUE && FPGA_STATUS==TRUE
Priority    = 0
Arguments   = TCargador /home/condor/compartido/animo
/home/condor/compartido/animo2.txt
Universe    = java
Output            = result/out/cargador.out
Log         = result/log/cargador.log
transfer_files    = ONEXIT
Queue
```

Donde como se puede observar, se manda el cargador con dos argumentos, el primero se supone que es el bitmap, y el segundo es el fichero con los datos de configuración. Para poder hacer la prueba de la correcta comunicación con el simulador el contenido del fichero /home/condor/compartido/animo es:

```
VENGA QUE SI FUNCIONA ESTO YA QUEDA POCO!!
```

Y el contenido del fichero /home/condor/compartido/animo2.txt es:

```
Q SI Q SI Q SI!!
```

Se puede observar también que en los requirements se ha puesto que es necesario que la máquina tenga tarjeta FPGA, y que su estado esté a cierto (esté libre).

Lo primero que tenemos que hacer es arrancar Condor en todas las máquinas del pool y arrancar los simuladores:

```
Microsoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\GHADIR>simulador_on

C:\Documents and Settings\GHADIR>C:\jdk1.2.2\bin\java -cp
c:\Condor\bin\ TSimula
dor
Esperando al cargador...
```

El simulador se queda esperando a que el cargador se comunique con él.

Inmediatamente después de realizar el submit del fichero de submit descrito anteriormente consultamos el estado del pool, y el resultado fue el siguiente:

```
C:\Documents and Settings\GHADIR\Mis documentos\Submits>condor_q


--   Submitter:   ghadir2.dacya.ucm.es   :   <147.96.81.68:1031>   :
ghadir2.dacya.ucm.es
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
  27.0    GHADIR            6/28 16:54    0+00:00:00 I   0    0.0   java
TCargador \\G

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

El trabajo está en estado idle, aun no ha pasado a ejecutarse.

Se vuelve a consultar el estado del pool un poco más tarde y el resultado obtenido es el siguiente:

```
C:\Documents and Settings\GHADIR\Mis documentos\Submits>condor_q


-- Submitter: ghadir2.dacya.ucm.es : <147.96.81.68:1031> :
ghadir2.dacya.ucm.es
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
  27.0   GHADIR          6/28 16:54   0+00:00:02 R  0   0.0  java
TCargador \\G

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

Ahora ya ha pasado a ejecutarse. Debería ejecutarse en ghadir1, puesto que es la única máquina del pool que cumple los requirements. Efectivamente se ejecuta en ghadir1. Esto lo podemos ver en el fichero de log que se ha generado:

```
000 (027.000.000) 06/28 16:54:18 Job submitted from host:
<147.96.81.68:1031>
...
```

```
001 (027.000.000) 06/28 16:54:24 Job executing on host:
<147.96.81.67:1029>
...
006 (027.000.000) 06/28 16:54:32 Image size of job updated: 6440
...
005 (027.000.000) 06/28 16:54:33 Job terminated.
      (1) Normal termination (return value 0)
      Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
      Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
      Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
      Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
60  -  Run Bytes Sent By Job
1986  -  Run Bytes Received By Job
60  -  Total Bytes Sent By Job
1986  -  Total Bytes Received By Job
...
```

El simulador que debe de estar ejecutándose muestra en el terminal la siguiente información:

```
cmd /c condor_set_var FPGA_STATUS FALSE
El comando cmd /c condor_set_var FPGA_STATUS FALSE ha terminado con
c+¦digo 0
\\Ghadir2\Submits\animo
\\Ghadir2\Submits\animo2.txt
null
Ejecutando Job...
```

Donde se puede observar que le ha llegado el trabajo y el cargador se ha comunicado satisfactoriamente con el simulador.

Si consultamos en este momento la variable FPGA_STATUS nos dice que está a FALSE porque la FPGA ahora mismo está ocupada ejecutando un trabajo. Se puede observar en el terminal que se ha ejecutado el comando `cmd /c condor_set_var FPGA_STATUS FALSE` para poner esa variable a falso. Se muestra por pantalla también las rutas del bitmap y de los datos de configuración que se encuentran en una carpeta compartida que se encuentra en la máquina Ghadir2.

Justo después de mostrar este mensaje ha editado con el bloc de notas dos ficheros, uno con contenido:

```
VENGA QUE SI FUNCIONA ESTO YA QUEDA POCO!!
```

y el otro con contenido:

```
Q SI Q SI Q SI!!
```

Con esto comprobamos que el simulador puede acceder a los ficheros cuya ruta le manda el cargador vía socket.

Una vez cerrados los ficheros, significará que el trabajo ha terminado de ejecutarse en la FPGA, y por tanto si se vuelve a consultar la variable PFGA_STATUS esta vez nos dice que es igual a TRUE.

Evidentemente se han realizado muchísimas más pruebas que estas, se han seleccionado las más representativas para entender el funcionamiento de condor y de nuestra aplicación.

11. BIBLIOGRAFÍA

11.1. REFERENCIADA

[1]. Página Web del proyecto de investigación GHADIR "http://www.ucm.es/info/ghadir/"

[2]. K. Compton, S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software", ACM Computing Surveys, Vol. 34, No. 2. pp. 171-210. June 2002.

[3]. O. F. Diessel, G. Wigley, "Opportunities for Operating Systems Research in Reconfigurable Computing", Technical report ACRC-99-018. Advanced Computing Research Centre, School of Computer and Information Science, University of South Australia, 1999.

[4]. Tabero J, Septién J, Mecha H, Mozos D., "Gestión de Hardware 2D Multitarea en FPGAs Dinámicamente Reconfigurables basado en Listas de Vértices", Instituto Nacional de Técnica Aeroespacial, Madrid & Dept. de Arquitectura de Computadores, UCM, Madrid, 2003.

[5]. A.V. Staicu, J.R. Radzikowski, K. Gaj, N. Alexandridis, and T. El-Ghazawi, "Effective Use of Networked Reconfigurable Resources", 2001 MAPLD International Conference, Laurel, Maryland, Sep. 2001.

11.2. UTILIZADA

Página Web del proyecto de investigación GHADIR "http://www.ucm.es/info/ghadir/"

Condor Team, "Condor Version 6.4.7 Manual University of Wisconsin-Madison", 7 de febrero de 2003

K. Compton, S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software", ACM Computing Surveys, Vol. 34, No. 2. pp. 171-210. June 2002.

O. F. Diessel, G. Wigley, "Opportunities for Operating Systems Research in Reconfigurable Computing", Technical report ACRC-99-018. Advanced Computing Research Centre, School of Computer and Information Science, University of South Australia, 1999.

Tabero J, Septién J, Mecha H, Mozos D. "Gestión de Hardware 2D Multitarea en FPGAs Dinámicamente Reconfigurables basado en Listas de Vértices" Instituto Nacional de Técnica Aeroespacial, Madrid & Dept. de Arquitectura de Computadores, UCM, Madrid, 2003.

A.V. Staicu, J.R. Radzikowski, K. Gaj, N. Alexandridis, and T. El-Ghazawi, "Effective Use of Networked Reconfigurable Resources", 2001 MAPLD International Conference, Laurel, Maryland, Sep. 2001.

Capítulo 11 de "Grid Computing – Making the Global Infraestructura a Reality." Edited by F. Berman, A. Hey and G. Fox. 2003 John Wiley & Sons, Ltd.

AUTORES

Ignacio José Horrillo Sancho
Germán Hurtado Martín
Ana María Vélez Villanueva