PLATAFORMA COMPUTING@HOME

Memoria del Proyecto Final de Carrera
de Ingeniería en Informática
realizado por
Carlos Moreno Losada
y dirigido por
Miquel Àngel Senar Rosell
Bellaterra, 13 de Junio de 2008

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

El abajo firmante, Miquel Àngel Senar Rosell

Profesor/a de la Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el trabajo al que corresponde esta memoria ha sido realizado bajo su dirección por Carlos Moreno Losada

Y para que conste firma la presente.

Firmado:

Bellaterra, 13 de Junio de 2008

Dedicado a mi familia por su apoyo incondicional,
a Miquel Àngel por su ayuda inestimable
y a Eduardo por creer en mi.
Para María, porque sin ella nada de esto sería posible.
Nunca os lo podré agradecer lo suficiente.

Carlos Moreno Losada.

Capítulo 1 - Introducción

Hace unas décadas, los científicos utilizaban grandes supercomputadoras (o *mainframes*) para procesar tareas computacionales. Aquellos superordenadores no tenían un gran rendimiento, pues ejecutaban los trabajos secuencialmente y los científicos tenían que esperar su turno para poder utilizar los recursos. Cada uno de ellos disponía de una cantidad de tiempo determinado para utilizar la potencia de cálculo del superordenador. Tiempo más tarde se pudieron utilizar entornos multiusuarios, en los que no hacía falta esperar un turno para la utilización del superordenador, pero los recursos del superordenador eran limitados y muchas veces los procesos quedaban paralizados hasta que otros procesos con mayor prioridad finalizaban. Estas supercomputadoras tenían un coste elevado y no se rentabilizaban lo necesario para paliar el coste, pero por otro lado éstas estaban utilizadas cerca del 100%, o sobreutilizadas.

Nuevas generaciones de ordenadores personales con renovados diseños y prestaciones que mejoraban y aumentaban generación tras generación, aparecieron en el mercado y por cuestiones de privacidad, rapidez y reducción de costes, los

usuarios pasaron de utilizar los *mainframes* a los ordenadores personales. Mientras los costes habían sido reducidos enormemente y los usuarios estaban cada vez más satisfechos con su ordenador personal, los recursos habían sido distribuidos y la capacidad de cómputo global había decrecido dramáticamente, ya que cada ordenador personal suponía una ínfima parte de procesamiento comparado con los superordenadores anteriormente utilizados. Además en los ordenadores personales se perdían grandes cantidades de ciclos de computación ociosos sin realizar trabajo alguno, que si se sumaran en un objetivo común podrían aumentar la capacidad de cómputo.

Cada vez los problemas científicos a resolver se vuelven más complejos y requieren de una gran capacidad de cómputo. Actualmente, un ordenador en solitario no es capaz de poder resolver este tipo de problemas, pero la unión de varios de estos ordenadores trabajando sobre partes estructuradas del problema pueden generar resultados útiles.

Para poder realizar estas tareas se han creado sistemas de computación distribuida. Estos sistemas permiten la distribución de tareas entre los diferentes recursos que están conectados al sistema para ejecutarlas.

En este proyecto veremos dos sistemas de computación distribuida muy diferentes entre ellos: Condor y BOINC, pero que trabajando conjuntamente pueden alcanzar objetivos comunes.

1.1 - Antecedentes

Diversos proyectos científicos actuales necesitan de una alta cantidad de recursos para ejecutar los trabajos creados en el proyecto. A menudo, estos trabajos necesitan ser procesados durante días o incluso varias semanas, y son necesarios sistemas que permitan controlar estos procesos durante el largo tiempo que se están ejecutando. Estos entornos que son capaces de gestionar una gran cantidad de

recursos ejecutados durante un largo lapso de tiempo son llamados *High Throughput Computing* (HTC).

En contraposición, encontramos los entornos *High Performance Computing* (HPC), que gestionan una gran cantidad de recursos para ejecutar procesos durante tiempos cortos. Su interés reside en ejecutar el máximo de procesos lo más rápido posible, mientras que los entornos HTC son más proclives a buscar el máximo número de aplicaciones ejecutadas durante un intervalo de tiempo largo.
El punto clave que los entornos HTC buscan es la eficiente gestión de los recursos disponibles, intentando ejecutar los procesos durante el tiempo que fuera necesario para llegar a la finalización de los trabajos y obtener resultados.

Condor es un software que crea un entorno HTC para la ejecución de trabajos. Puede gestionar *clusters* de ordenadores dedicados, comunicados por una red de área local, utilizando el máximo rendimiento de la capacidad de cómputo disponible que ofrecen los ordenadores.

Condor fue diseñado en la Universidad de Wisconsin-Madison y es un sistema ampliamente utilizado en diversos lugares para gestionar *clusters* de ordenadores. Como ejemplo, tenemos la misma Universidad de Wisconsin-Madison, donde reside el primer sistema Condor implementado y que gestiona más de 1.000 ordenadores que ejecutan proyectos científicos constantemente.

Dentro de los proyectos científicos existen trabajos que requieren ejecutarse varios cientos de veces durante un periodo amplio de tiempo. Este tipo de proyectos requieren entornos de computación HTC como Condor, que puede ejecutar un trabajo cientos o miles de veces, con cientos y miles de conjuntos de datos a la vez, ahorrando tiempo al lanzarlo. Con una sola orden todos los trabajos son enviados a Condor y dependiendo del número de ordenadores que pueden ejecutarlos, los procesos quedarán en cola esperando donde ejecutarse. El tiempo de ejecución y de espera puede ser extenso, pero Condor se encargará de la gestión de los trabajos hasta su finalización.

Condor proporciona una gestión de recursos potente, gracias al *matchmaking* de recursos que realiza. Este es el éxito de un entorno HTC como Condor. Éste hace coincidir los productores de recursos con los consumidores de los mismos , adaptando los requisitos al máximo.

Otros sistemas de gestión de recursos utilizan colas de trabajos diferentes dependiendo de las características especiales asignadas a cada cola y son ellos mismos los que añaden propiedades a las colas de trabajos al instante, generando confusión al usuario sobre qué cola se debe usar, por los cambios realizados en las propiedades, para satisfacer ciertas demandas de otros usuarios.

Condor, en cambio, implementa el sistema *ClassAd*, que proporciona un diseño claro que simplifica el envío de trabajos por parte del usuario y la sencillez para encontrar el ordenador adecuado para ejecutar el proceso. Los *ClassAds* son un método eficaz para poder planear la ejecución de procesos como, por ejemplo, la implementación de diversos procesos capaces de diseñar grafos acíclicos de ejecución.

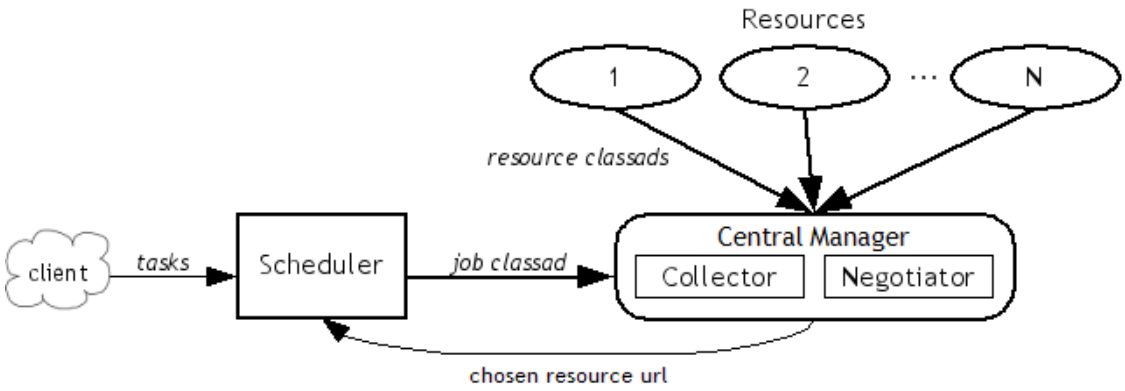

Figura 1 - Representación del sistema de *matchmaking*

Condor puede ser usado para construir entornos de computación de estilo *Grid* que atraviese los límites administrativos y proporciona herramientas para incorporar

estos sistemas, como puede ser el caso de la tecnología *flocking*, que permite que diversos sistemas Condor trabajen conjuntamente, o la interacción con Globus.

Otro sistema de computación es BOINC (*Berkeley Open Infrastructure Network Computing*) y es una infraestructura de computación distribuida y voluntaria. El sistema permite que un usuario pueda instalarse un cliente fácilmente y se registre en proyectos que utilizan la infraestructura BOINC para colaborar con ellos.

La instalación del cliente BOINC es sencilla de realizar sea cual sea la arquitectura y el sistema operativo usado, ya que BOINC se encuentra disponible para una gran variedad de sistemas operativos y arquitecturas de computadores. Cada uno tiene su propio sistema de instalación, pero la infraestructura BOINC está pensada para que el público en general ayude en los proyectos y por lo tanto se facilitan los procesos para poder colaborar. Esta colaboración se consigue atrayendo a los usuarios con proyectos de gran interés general y de añadidos como salvapantallas o *skins* que permiten adaptar el cliente según los gustos de cada usuario.

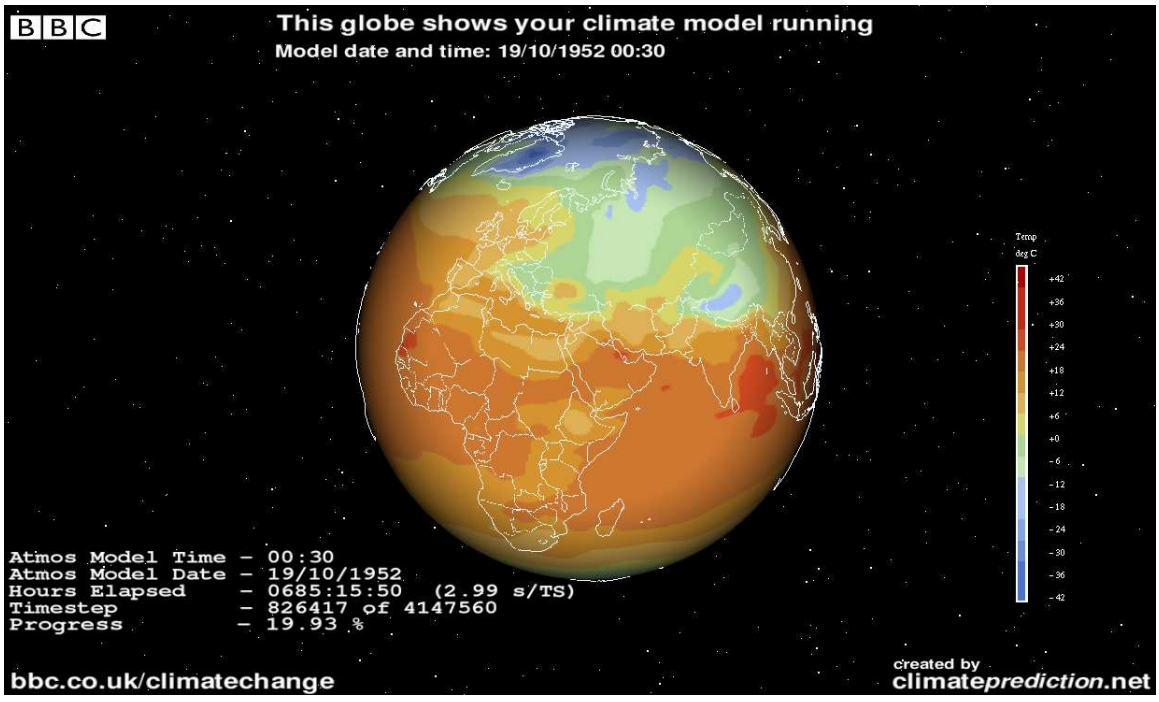

Figura 2 - Salvapantallas del proyecto ClimatePrediction.net

Una vez instalado el cliente y registrado el usuario en el proyecto que ha elegido, se utiliza el tiempo que el ordenador está ocioso para ejecutar el proyecto en el que se ha suscrito. Los proyectos suelen ser de investigación científica que requieren de sistemas de computación masiva y buscan la colaboración de voluntarios para poder ejecutar sus tareas.

Se trata de un sistema voluntario, ya que el usuario dona sus recursos voluntariamente para ejecutar las tareas de los proyectos a los que se ha registrado, donando su tiempo ocioso de procesador para ejecutar los procesos de los proyectos BOINC. Si el proyecto no funcionara por algún motivo, los recursos que el usuario ha asignado en su ordenador para ese proyecto se repartirían entre el resto de proyectos.

Proyectos BOINC son, por ejemplo:

- El proyecto SETI, el cual se dedica, básicamente, a la búsqueda de vida extraterrestre analizando las señales que la NASA capta en sus antenas.
- El proyecto FightAIDS@home, que centra su trabajo en el descubrimiento de diferentes partículas que ayuden a la lucha contra el SIDA.
- El proyecto CancerResearch@home, que usa la computación voluntaria distribuida para la investigación contra el cáncer.
- El proyecto Rossetta. Este proyecto investiga proteínas en 3D que puedan encontrar una solución a enfermedades mundiales como por ejemplo el cáncer o el Alzheimer.
- El proyecto Einstein@home investiga *pulsars* en las señales recibidas en los laboratorios que estudian las estrellas.
- El proyecto ClimatePrediction.net estudia el cambio climático añadiendo miles de modificaciones para comprobar como se comporta el sistema.

Un mismo usuario mediante su cliente BOINC instalado puede colaborar con diversos proyectos, no teniendo que restringirse solamente a uno, compartiendo el tiempo de proceso entre los diferentes proyectos a los que el usuario está suscrito.

1.2 - Objetivos

Los sistemas de computación distribuida son diversos y cada uno aporta una solución a un tipo de problema concreto.
Condor actúa sobre *clusters* de ordenadores, proporcionando sistemas generalistas de altas funcionalidades, gestionando de manera eficiente y fiable los procesos a ejecutar en ordenadores que dedican su tiempo a la ejecución masiva de proyectos.
BOINC, en cambio, realiza una computación intensiva de proyectos creados atractivamente para el público, creados con el fin de ser resueltos mediante la computación voluntaria en una infraestructura sencilla y configurable donde cualquier usuario doméstico puede colaborar.
Como se puede ver, cada sistema está diseñado para obtener la misma finalidad pero de diferentes formas, desde diferentes puntos de vista.

En este proyecto se explorarán las posibilidades que existen de aproximar ambos sistemas y conseguir que los sistemas de gestión de *clusters* HTC, como Condor, tengan la posibilidad de incluir sistemas de computación voluntaria como BOINC, para poder ejecutar trabajos controlados por un gestor de recursos fiable y seguro en ordenadores voluntarios de usuarios domésticos, los cuales donan sus ciclos de CPU desinteresadamente mediante una infraestructura flexible y llamativa.

Iniciamos parte de la idea con un conjunto de recursos Condor, donde los usuarios pueden enviar sus tareas a ejecutar y el sistema Condor las gestiona asignándolas a los diferentes recursos.
BOINC, por su parte, dispone de su infraestructura en la que las tareas del proyecto a ejecutar se reparten entre los usuarios registrados en el proyecto.
Se busca la opción de poder enviar trabajos Condor que puedan ser ejecutados en ordenadores externos al ámbito del gestor y que colaboren en un proyecto BOINC creado para este fin.

Paralelamente se conseguirá aumentar el número de recursos del *pool* para poder ejecutar más tareas a la vez, permitiendo que los usuarios reciban la solución de sus

múltiples tareas con mayor celeridad. Una segunda vía de pensamiento es la opción de que diversos usuarios requieran de un número de recursos para ejecutar sus tareas que el *pool* no las pudiera ofrecer por la limitación de recursos. Esta estructura nos limita a ejecutar tareas que requieran el número máximo de recursos que dispone el *pool.*

En buena parte se quiere incrementar el número de recursos, pero este incremento no es sencillo de realizar. Dentro de las posibles soluciones se sabe que se realizará un incremento dinámico de recursos, pues se conoce que la computación voluntaria dispone de un contratiempo como es la volatilidad de las conexiones de los ordenadores al sistema. Estas conexiones dinámicas se deben tener en cuenta, ya que si un proceso necesita de una fiabilidad del 100%, con este método no se podrá ofrecer por su volatibilidad.

Se intentará encontrar una solución para el incremento de recursos del *pool* de Condor, añadiendo máquinas externas voluntarias, pero existen ciertos problemas o requerimientos a cumplir:

- Un problema que pudiera surgir sería que el sistema añadiera recursos externos y éstos pudieran generar problemas de seguridad al no encontrarse en un entorno seguro. En principio, el sistema no está limitado a un conjunto de usuarios restringido y los entornos que se pueden añadir pudieran ser muy diversos. Se debe tener en cuenta la seguridad del sistema, eliminando aquellos riesgos potenciales que se puedan detectar, minimizando el efecto de los riesgos que surgieran y que no hubieran sido detectados. En cierta forma deberíamos cuidar tanto el sistema como el entorno en el que se encuentra el sistema, impidiendo que dejase de funcionar por completo si se produjera algún problema de seguridad.

- Otro requisito impuesto para la solución del problema es la facilidad, accesibilidad, y transparencia que el usuario externo debe tener para poder colaborar en la ejecución de las tareas. El sistema de adición de recursos al proyecto debería ser accesible para el mayor número de público interesado,

fácil de instalar y de poner en funcionamiento, proporcionando una transparencia al usuario en el momento de la ejecución para no entorpecer el trabajo del mismo. Como se trata de un sistema dinámico, éste debe ser flexible tanto a la hora de añadir recursos como de sustraerlos.

- Un sistema de estas características debería poder aportar una fiabilidad para que las tareas se ejecutaran en un tiempo razonablemente finito y que se pudiera obtener el resultado de tal ejecución. Sin este sistema de fiabilidad podría darse el caso que las tareas fueran enviadas a recursos que desaparecieran del *pool* y que nunca más pudieran volver a conectarse. El usuario emisor de la tarea nunca recibiría el resultado, podría incluso retardar otras tareas que necesitasen de este resultado o hacerlas fracasar por no ejecutarse en un tiempo dado. Algún mecanismo implementado debería evitar estas situaciones, aunque si bien podría ser tolerable algún fallo que algún recurso pudiera generar, restableciendo la ejecución en otro recurso. Esta tolerancia y transparencia de fallo debería ser inapreciable por parte del usuario emisor de la tarea.

- En algunas ocasiones la apertura de los sistemas a un mayor número de usuarios puede provocar que los componentes centrales del sistema queden bloqueados por saturación. Este es un problema típico de las redes y de los servidores que gestionan varios cientos o miles de recursos. Como el sistema no indica un número máximo de recursos que se pueden gestionar, deberemos limitar el acceso a recursos que pretendan añadirse al sistema, para impedir la saturación del mismo y sus componentes.

- Otro aspecto importante es la gestión concurrente y las comunicaciones entre las diferentes tareas. Para facilitar el proyecto se realizarán pruebas en las que las tareas carecen de comunicación alguna con otras semejantes u otros recursos ajenos al sistema. Esto nos proporcionará un primer paso que indique la viabilidad del proyecto.

1.3 - Gestión del proyecto

1.3.1 - Planificación de tiempos

La planificación de tiempos está marcada por las tareas a desarrollar y los hitos a conseguir. Las tareas se dividen en lo que se llama la *Work Breakdown Structure* (WBS) y especifica de la manera más concreta posible las tareas a realizar y la distribución de tiempos dependiendo de la interacción que tengan entre ellas. Como resultado de todo este estudio, podemos ver el diagrama de Gantt resultante que nos ayuda a conocer el estado del proyecto en un momento determinado, pudiendo decidir si son necesarias acciones correctivas en el proyecto para poder llevarlo a cabo en los tiempos determinados.

Se mostrará y describirá brevemente cada una de las fases que hemos generado para la planificación del proyecto y se pasará a la visualización del diagrama de Gantt correspondiente al proyecto.

1.3.2 - WBS del proyecto

- *Análisis del problema*: Este es un apartado previo a la implementación del proyecto. Este análisis es necesario para idear la solución más óptima al problema que tenemos.
- *Instalación Condor*: Se trata de la instalación del sistema Condor del *testbed* que nos servirá para realizar las pruebas de funcionamiento. También incluye las pruebas realizadas sobre el mismo sistema que determinan su funcionamiento correcto.
    - *Configuración servidor Condor*
    - *Configuración cliente Condor*
    - *Creación aplicación prueba Condor*
    - *Pruebas funcionamiento aplicación Condor*
    - *Creación cortafuegos para Condor*
- *Instalación BOINC*: El *testbed* necesita también de un pequeño sistema BOINC que ayude a realizar las tareas de prueba. Se añaden las pruebas a

realizar para conocer el funcionamiento del sistema y la posterior comprobación del mismo.

  - *Configuración servidor BOINC*
  - *Configuración cliente BOINC*
  - *Creación proyecto prueba BOINC*
  - *Pruebas funcionamiento proyecto BOINC*
  - *Creación cortafuegos para BOINC*

- *Condor + BOINC*: La unión de estos dos sistemas nos proporcionará la realización del proyecto. Se debe configurar Condor encapsulándolo como tarea a ejecutar de BOINC. Se creará un proyecto de BOINC con la tarea realizada y se realizarán las pruebas anteriores que puedan ayudar a la comprobación.
  - *Creación proyecto Condor en BOINC*
    - *Configuración condor_config*
    - *Encapsulado BOINC de paquete Condor*
  - *Creación de proyectos BOINC*
  - *Pruebas funcionamiento proyecto*
  - *Creación cortafuegos para proyecto*
- *Documentación*: Se realizará la escritura de la documentación pertinente al proyecto.
- *Presentación*: Se realizará la presentación pública del proyecto.

1.3.3 - Diagrama de Gantt

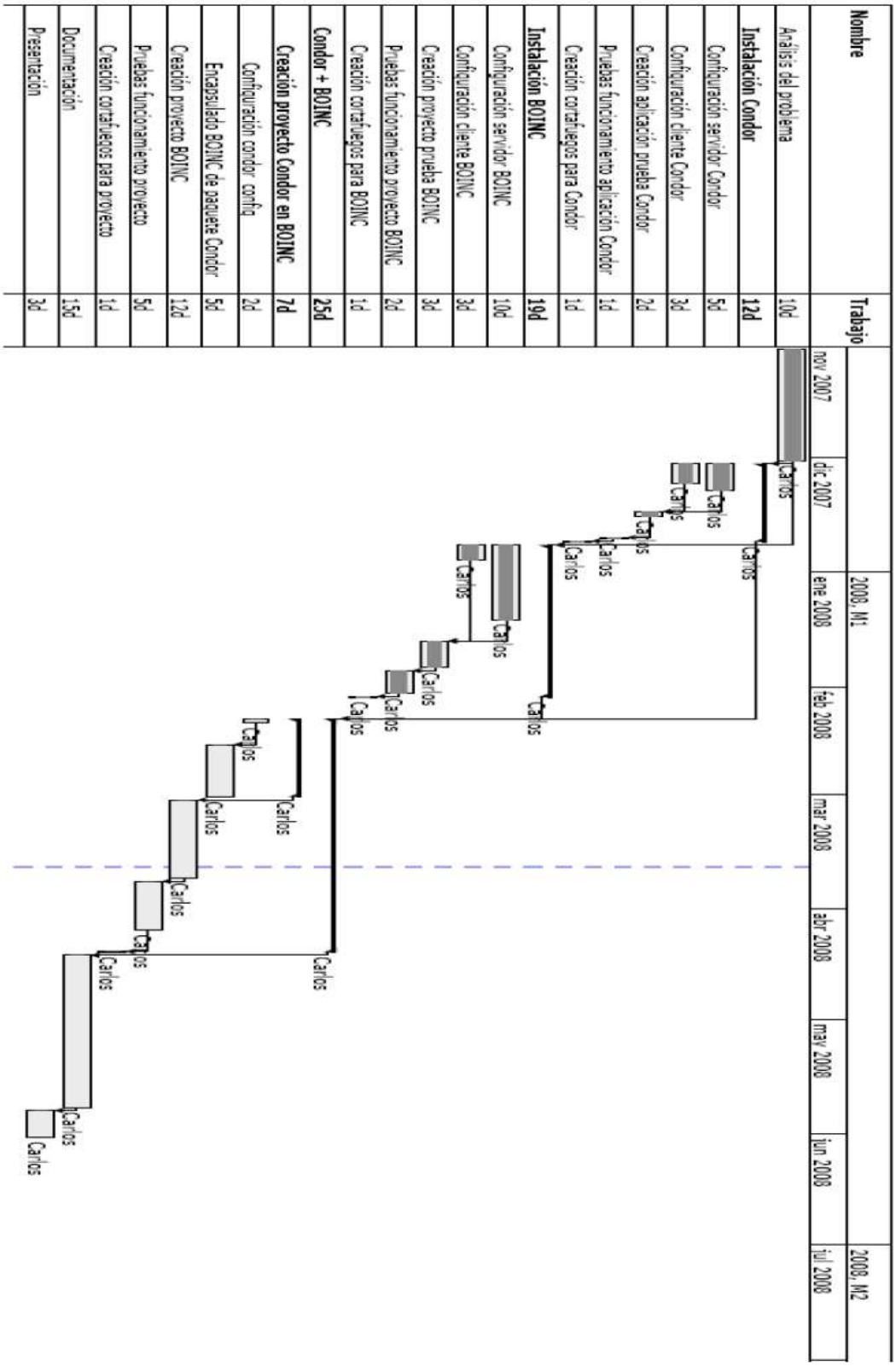

Figura 1 - Diagrama de Gantt

1.4 - Estructura de la memoria

La estructura de la memoria se repasará en esta sección, los diferentes capítulos que veremos y el contenido de los mismos.

El capítulo 2 trata los fundamentos teóricos necesarios y los métodos utilizados para entender y comprender el trabajo realizado. Estos conceptos engloban las bases teóricas del proyecto.
Los puntos a ver son los sistemas distribuidos y su clasificación, el sistema Condor, un sistema HTC ampliamente utilizado en diversos proyectos, y el *middleware* BOINC, una infraestructura creada para la computación voluntaria de proyectos.

El capítulo tercero está dedicado a los problemas surgidos durante la ejecución del proyecto y se verán las soluciones tomadas en cada momento para cada problema. De esta forma se repasará el cumplimiento de los requisitos iniciales del proyecto.
Se han contemplado las arquitecturas de los sistemas en los que está basado el proyecto, y se mostrará la arquitectura del sistema y la implementación desarrollada del proyecto.

El cuarto capítulo mostrará la instalación de los entornos de pruebas que fueron necesarios para poder ejecutar el proyecto y la experimentación realizada en ellos. En este apartado se dará una solución pragmática a las implementaciones anteriormente explicadas.

El capítulo 5 consta de los apartados correspondientes a las conclusiones obtenidas en el trabajo realizado y las líneas futuras que el proyecto abre a nuevas extensiones. Éstas son importantes para conocer el alcance obtenido con el proyecto respecto a las posibilidades del conjunto del trabajo.

Capítulo 2 - Fundamentos teóricos

La computación distribuida es un modelo de computación que ayuda a resolver problemas de computación masiva y que utiliza, en su gran medida, un número elevado de ordenadores para resolver problemas que requieren un alto índice de computación.

El proyecto desarrollado está basado en sistemas de computación distribuida. Éstos son un modelo utilizado en diversas áreas para resolver grandes retos que requieren un alto índice de computación, como la física, la biología molecular, la medicina o la meteorología.
Estos grandes retos pueden ser, por ejemplo, el plegamiento de proteínas, el descubrimiento de nuevos medicamentos, la simulación de terremotos o inundaciones, la simulación financiera o el estudio de un tema habitual hoy en día, la creación de modelos climáticos, realizando cálculos de sistemas que estudian modelos de los posibles efectos del calentamiento global del planeta.

Todos estos ejemplos tienen en común la necesidad de utilizar computación masiva, la cual puede ser llevada a cabo de diferentes formas que veremos a continuación.

2.1 - Sistemas Distribuidos

Cuando se piensa en la ejecución de grandes proyectos se suele pensar en un superordenador que ejecuta incesantemente todos los trabajos. Ciertamente, una de las formas de realizar computación masiva es el uso de un supercomputador que pueda procesar las tareas suministradas por el proyecto. Afortunadamente no es la única manera de llevar a cabo esta tarea. Cuando este supercomputador no se puede hacer cargo del trabajo o si no se dispone de un supercomputador por los costes que éste implica, se utilizan sistemas distribuidos que ayudan a realizar los cálculos requeridos para resolver los trabajos.
Este es el objetivo por el cual la computación distribuida fue diseñada, permitiendo además de la ejecución de estas tareas, la flexibilidad para poder trabajar en múltiples problemas más pequeños que ayuden colateralmente a resolver grandes problemas y obtener un rendimiento mayor del conjunto global que compone el sistema.

Dentro de la computación distribuida existen diferentes tipos de sistemas que requieren de sus necesidades. Algunos de ellos son:

- La existencia de servicios web XML nos proporcionan servicios y aplicaciones que pueden ser accedidos para utilizar entornos distribuidos. Estos servicios web XML están regidos por un estándar llamado *Open Grid Services Architecture* (OGSA). Este estándar es el utilizado por Globus Toolkit en su versión 3.0.
- La computación de ciclos redundantes o también llamada computación zombie está compuesta por un servidor o grupo de servidores que distribuyen tareas para ser procesadas entre los diferentes sistemas voluntarios que se han adjuntado para colaborar en la ejecución del

proyecto. Los ordenadores voluntarios ejecutan estas tareas cuando se encuentran ociosos, no interfiriendo en la ejecución y uso del ordenador por parte del usuario. De esta manera, se dona a la ejecución de los proyectos los ciclos redundantes, aprovechando al máximo la capacidad de procesamiento del ordenador.

- Los *clusters* de ordenadores son un conjunto de ordenadores de bajo coste relativo, unidos entre si por una red local de alta velocidad. Todo el *cluster* de ordenadores suele disponer del mismo sistema operativo y de un software que permite la ejecución de tareas distribuidas y establece las comunicaciones entre los diferentes elementos del *cluster*. Otro dato típico de los *clusters* de ordenadores suele ser la existencia de un único sistema de almacenamiento compartido.
- Los sistemas *Grid* son un paradigma de la computación distribuida aparecido en las últimas décadas. Estos sistemas disponen de un número indeterminado de ordenadores dedicados, que funcionan como si se trataran de uno único. Esta unión se realiza de manera transparente, generando a la vista del usuario un único recurso que pueda ejecutar las tareas requeridas, aunque estos ordenadores se encuentren ubicados en diferentes lugares geográficos. Para llevar a cabo estas funciones se suele utilizar un software determinado que permite las comunicaciones entre ordenadores, la gestión de almacenamiento de datos y el envío de trabajos, entre otras tareas. Uno de los softwares más famosos para crear estos sistemas es Globus. Globus es un *middleware* con el que se puede crear un sistema G*rid*. Permite gestionar y descubrir los recursos, así como el control del almacenamiento de datos.

La existencia de diferencias entre los sistemas distribuidos permiten clasificarlos de diversas formas, entre ellas encontramos la homogeneidad de los sistemas o la localización de los mismos.

Por ejemplo, los sistemas llamados *Single System Image* (SSI), donde todos los recursos computacionales disponen del mismo sistema operativo diseñado expresamente para permitir trabajar en un entorno distribuido. Ejemplos de estos SSI son DragonFly BSD, Mosix/Open Mosix o Open SSI. En contraposición

encontramos sistemas *Grid* donde la heterogeneidad de los sistemas es muy alta, pudiendo encontrarse dentro de un mismo sistema *Grid* diferentes clases de sistemas operativos.
Otro ejemplo de clasificación es la localización de los elementos, como los *clusters* de ordenadores que se encuentran localizados dentro de la misma ubicación, conectados mediante una red local de alta velocidad que conecta todos los recursos, mientras que en un sistema *Grid* los ordenadores se encuentran ubicados en diferentes lugares del mundo, pudiendo comunicarse a través de Internet.
Estas clasificaciones se pueden ampliar con la visión que el usuario tenga del sistema. Un *cluster*, por ejemplo, utiliza todos los ordenadores buscando mejorar el rendimiento de todos ellos, mientras que en los sistemas *Grid* el englobe total de ordenadores quiere ofrecer la visión al usuario de un superordenador, suplantando los elementos dispersos.
Otro aspecto importante en los sistemas distribuidos es la fiabilidad del sistema. Se puede dar una alta fiabilidad, como en los sistemas *Grid*, donde el mal funcionamiento de un nodo o recurso no implica el paro del sistema, pudiendo servirse de otros recursos del *Grid*, o una fiabilidad baja, que puede ser, por ejemplo, un *cluster* de ordenadores donde el mal funcionamiento de un recurso puede provocar la parada del sistema completo.

Diversa literatura incluye los sistemas oportunistas de computación voluntaria como una clase de sistemas de computación distribuida *Grid*, aunque la computación voluntaria difiere de la computación Grid, porque esta última involucra la gestión de recursos dentro y entre organizaciones virtuales. En cambio la computación voluntaria no requiere de esta organización. Se podría considerar un sistema Grid cuando se organiza los recursos por proyectos, realizando una similitud con las organizaciones virtuales. Por este motivo veremos los sistemas *Grid* y posteriormente los sistemas de computación voluntaria.

El término *Grid* fue acuñado por Ian Foster y Carl Kesselmans en los principios de los años 90 en un seminario llamado *The Grid: BluePrint for a new Computing Infrastructure*. Este seminario desembocó en un libro con el mismo nombre.

Los autores buscaban una metáfora para comparar la facilidad de acceso de este método de computación a la red de suministro eléctrico. De aquí se extrajo el vocablo inglés *Grid* que sirve para denominar estos sistemas de computación.

El término *Grid Computing* dispone de dos acepciones en el diccionario tecnológico que engloban dos subcategorías de sistemas distribuidos:

- *Grid* es la computación online o el almacenamiento ofrecido como servicio soportado por un conjunto de recursos distribuidos, conocidos también como utilidad de computación , computación bajo demanda o *cluster* computacional. Existen *Grids* de almacenamiento de datos que proveen el control para el almacenaje de grandes cantidades de datos que pueden ser compartidos. Estos *Grids* suelen ofrecer apoyo a los *Grids* computacionales.
- Los sistemas *Grids* buscan la creación de un “superordenador virtual” compuesto de una red de ordenadores conectados, ejecutando conjuntamente tareas de gran tamaño que no podrían ser ejecutadas por ordenadores en solitario. Estos sistemas se han utilizado para ayudar en el área de computación científica intensiva, matemática y resolver problemas académicos a través de la computación voluntaria. También se han aplicado sobre sistemas comerciales como por ejemplo la pronosticación económica, el descubrimiento de drogas y medicamentos o el procesamiento de datos en *back-office* para el soporte al comercio electrónico y servicios web.

En esta segunda definición encontramos una referencia a los sistemas de computación voluntaria, que son un tipo de sistemas de computación distribuida en la cual los propietarios de ordenadores, habitualmente domésticos, donan sus recursos informáticos (almacenamiento, procesamiento) voluntariamente para ejecutar tareas de uno o más proyectos.

El primer proyecto de computación voluntaria conocido fue el *Great Internet Mersenne Prime Search*, que se inició en enero de 1996. En años sucesivos surgieron otros proyectos como *Superweb*, *Popcorn*, *Charlotte* o *Bayanihan*.
El desarrollador de este último proyecto, *Bayanihan*, fue quien acuñó el término "computación voluntaria", Luis F.G. Sarmenta.

El software cliente de los primeros proyectos de computación voluntaria consistía en un simple programa que combinaba la computación distribuida y la infraestructura del sistema. Esta arquitectura monolítica no era flexible, ya que por ejemplo era difícil actualizar las versiones sin modificar la infraestructura.
Recientemente se han desarrollado sistemas *middleware* que proveen una infraestructura de computación distribuida independiente de la computación científica, por ejemplo:

- BOINC, desarrollado por la Universidad de California.
- XtremWeb, desarrollado por la Universidad Paris-South.
- Xgrid, desarrollado por Apple para Mac OS X.
- GridMP, desarrollado por United Devices para uso comercial.

La estructura básica que siguen estos sistemas es la de un programa cliente que se ejecuta en el ordenador voluntario. El usuario controla el progreso y la dedicación dada a la ejecución de tareas del proyecto y es el cliente instalado el que periódicamente contacta con el servidor del proyecto, entregando los resultados obtenidos y recibiendo nuevas tareas para ejecutar, si se da el caso. De esta manera, con el sistema *pull* se pueden evitar los posibles *firewalls* instalados en los ordenadores clientes que no tengan permitidas conexiones de entrada.

Los sistemas de computación voluntaria deben controlar algunos aspectos problemáticos.

⇒ La heterogeneidad de los sistemas que participan.
⇒ La disponibilidad esporádica de los recursos participantes.

- ⇒ La necesidad de la no interferencia del ordenador cuando sea el usuario quien realiza el uso.
- ⇒ El anonimato de los usuarios.
- ⇒ El sistema numérico de medida (créditos) para reconocer la cantidad de trabajo realizado por cada usuario.
- ⇒ La detección de resultados incorrectos.
- ⇒ La reclamación de créditos por resultados erróneos.

Para resolver estos problemas se utilizan soluciones como la “computación replicada”, en la que cada trabajo es realizado como mínimo por dos ordenadores y solamente los resultados son aceptados si ambos son similares o cercanos. Si esto ocurre, se asignan créditos a ambos usuarios por el trabajo realizado.

Por el lado cliente también existen problemas como el incremento del consumo eléctrico, ya que el ordenador consume más electricidad al estar activo que encontrándose en un estado ocioso. Se suelen desactivar funciones como “Suspender” o “Hibernar” en el ordenador para permitir la ejecución de estos procesos, con la consecuencia que el ordenador está continuamente ejecutando tareas.

Si la memoria RAM se convierte en una limitación entre todos los recursos, el rendimiento del ordenador decaerá al aumentar los fallos a caché y la paginación a disco. Para evitar estos problemas se suelen ejecutar las aplicaciones de los sistemas voluntarios con baja prioridad, lo que ayuda a aliviar la contención de la unidad de proceso. Estos efectos en el cliente pueden ser visibles o no, e incluso si son visibles el voluntario suele escoger seguir colaborando con los proyectos.

Otro problema es el decreciente rendimiento del ordenador si no está bien controlado el uso del ordenador durante la ejecución del proyecto. Se pueden definir aspectos críticos como la cantidad de disco a utilizar o los momentos de ejecución de las tareas. Una mala configuración de estas características puede suponer la saturación de los procesos del ordenador.

2.2 - Condor

Condor es un sistema de procesos por lotes especializado para gestionar trabajos de computación intensiva y proporciona un mecanismo de encolado de trabajos, políticas de programación, esquemas de prioridad y clasificación de recursos.

Los sistemas de procesos por lotes normalmente son ejecutados por sistemas dedicados, que pertenecen a una organización y tienen como único propósito la ejecución de trabajos. Las posibilidades de Condor son que puede enviar trabajos a ordenadores dedicados pero también puede enviarlos a ordenadores no dedicados, utilizando los momentos que el usuario no se encuentra utilizando el ordenador.

Condor es un software que crea un entorno HTC y puede gestionar *clusters* de ordenadores dedicados comunicados por una red de área local, utilizando el máximo rendimiento de la capacidad de cómputo disponible.

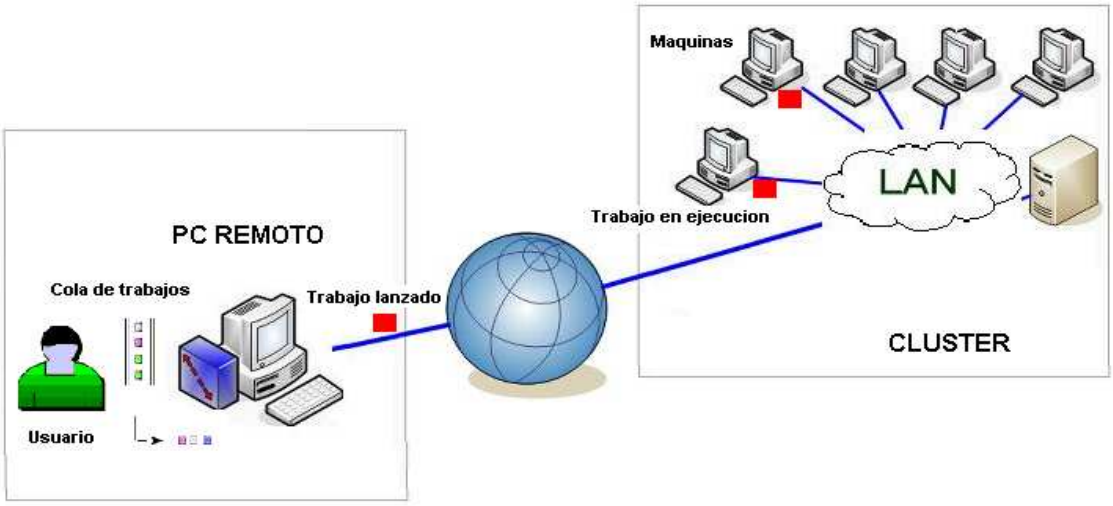

Figura 3 - Esquema de componentes de Condor

Los usuarios envían sus trabajos informáticos a Condor, y éste coloca los trabajos en una cola, según la política especificada, busca ordenadores disponibles en la red

que se adapten a los requisitos de los trabajos, los ejecuta y entonces informa al usuario sobre el resultado.

Condor proporciona mediante *matchmaking* de recursos una gestión de los mismos entre productores y consumidores de recursos. Este es el secreto del éxito de Condor que ha permitido que sea uno de los sistemas más ampliamente utilizado para este tipo de entornos.

2.2.1 - Características

Las características que hacen de Condor un sistema diferente y excepcional son las siguientes:

- *Checkpoint* y Migración → Cuando los trabajos son enlazados con las librerías de Condor, los usuarios pueden asegurar que sus trabajos serán posiblemente completados, incluso en los entornos cambiantes que Condor utiliza. Cuando un ordenador ejecutando un trabajo enviado se convierte en indisponible, el trabajo puede detenerse en ese instante guardando todo el estado actual. El trabajo puede continuar después de haberse migrado a otro ordenador. Condor realiza puntos de control periódicamente para, de alguna forma, salvaguardar el tiempo acumulado de computación en un trabajo, no perdiéndose todo el trabajo en caso de apagado del ordenador.
- Llamadas de sistemas remotas → Condor, mediante las llamadas del sistema remotas puede conservar el entorno de ejecución local en el entorno *Condor standard universe*. No es necesario preocuparse de la transferencia de archivos o de realizar *login* en el ordenador remoto, los trabajos se realizan como si fueran locales y es transparente el ordenador en el que se ejecuta el trabajo.
- No son necesarios cambios en el código fuente → Condor es capaz de ejecutar procesos interactivos, realizar puntos de control y migración transparentemente siempre y cuando reenlacemos el código fuente con las librerías de Condor al compilar el trabajo a ejecutar. Si no se proporciona

el entorno *Condor vanilla universe* para ejecutar aplicaciones que no se pueden reenlazar por la falta del código fuente.

- *Pools* de ordenadores operan conjuntos → *Flocking* es una característica de Condor que permite que trabajos enviados a un *pool* sean ejecutados en un segundo *pool* al que se tiene confianza. El mecanismo es flexible pudiendo configurar políticas en el segundo *pool* para conocer qué ordenadores y qué trabajos pueden ser ejecutados.
- Usuario de ejecución → En cuestiones de seguridad Condor no requiere una cuenta de usuario en los ordenadores destino donde se ejecutan los trabajos. Es capaz de utilizar cuentas generalistas como *nobody*, pero se debe configurar para este aspecto.
- Ordenación de trabajos → Se pueden ordenar las ejecuciones de los trabajos haciendo que cada uno de ellos sea un nodo dentro de un grafo acíclico. Se establecen dependencias en la ejecución de los trabajos según las expresadas en el grafo.
- Habilitación de *Grid Computing* → Utilizando la técnica de *Glidein* se permite enviar a ejecutar trabajos de Condor a ordenadores que pertenecen a un *Grid* ubicados alrededor del mundo. Estos recursos *Grid* son manejados por Globus.
- Percepción del propietario del ordenador → Los procesos del propietario de un ordenador que colabore con Condor tienen prioridad sobre el uso del ordenador. El propietario cede el tiempo no utilizado para ejecutar otros procesos como Condor. Es transparente y el usuario no debe realizar ninguna operación, se realiza automáticamente. La capacidad de detectar si el usuario de un ordenador está trabajando con él la realiza Condor y si existiera un trabajo ejecutándose en ese momento, Condor lo pararía de inmediato e intentaría realizar un *checkpoint* y migrar el trabajo a un ordenador diferente, el cual admitiera el trabajo. Condor retomaría el trabajo en el nuevo ordenador exactamente en el punto donde paró la ejecución.
- *ClassAds* → El mecanismo *ClassAds* proporciona un *framework* flexible y expresivo para realizar la mejor correspondencia entre las peticiones de recursos y las características de los recursos, indicadas por los trabajos

lanzados y por las políticas expresadas en los ordenadores del *pool*. Durante el proceso de *matchmaking,* Condor también considera diversas capas de valores prioritarios:

- Prioridad del usuario asignada al anuncio de petición del recurso.
- Prioridad del usuario que envía el anuncio.
- Deseos de los ordenadores en el *pool* para aceptar ciertos tipos de anuncios sobre otros.

Por el contrario, Condor también tiene limitaciones que no permiten realizar algunas funciones. Es importante conocerlas:

- Limitaciones en trabajos con *checkpoint* → No se pueden realizar puntos de control y migración de:
  - Trabajos con multiprocesos. Por ejemplo trabajos que utilicen las funciones *fork* o *exec*,
  - Comunicación entre procesos (*Inter-Process Communication*, *IPC*). Procesos que utilicen *pipes* o semáforos
  - Operaciones de red de larga duración. Los procesos que abran un *socket* no pueden dejarlo abierto durante mucho tiempo.
  - Utilización de las señales SIGUSRZ y SIGTSTP. Otras señales están permitidas, pero éstas Condor las utiliza internamente.
  - Procesos dormidos. Los procesos que utilicen funciones como *alarm* o *sleep*.
  - Utilización de múltiples *kernel-level threads*. Los *thread*s de usuario están permitidos, pero si son del kernel no.
  - Ficheros con *memory mapped*.
  - Bloqueo de ficheros. No se pueden utilizar funciones como *lock*.
  - Ficheros no abiertos de sólo escritura o lectura. Los ficheros abiertos de lectura y escritura pueden provocar fallos.
  - Ordenadores con poco espacio en disco. Es necesaria una cantidad de disco para guardar el estado de la tarea.
  - Procesos que lean o escriban en ficheros de mas de 2 GB.

- Seguridad → Condor proporciona sistemas y políticas de seguridad, pero no puede controlar que ciertos sistemas tengan problemas de seguridad.
- Los trabajos deben ser enlazados de nuevo → Para obtener las propiedades de checkpoint y de señales de sistema remotas, los trabajos deben ser *linkados* con las librerías de Condor.

2.2.2 - Arquitectura de Condor

El sistema de gestión de colas de trabajos Condor es una pieza fundamental del proyecto. Condor dispone, en su arquitectura básica, de un servidor central llamado *Central Manager* que es el encargado de realizar la gestión de recepción y envío de trabajos, así como la correspondencia entre trabajos y recursos. Se puede considerar el cerebro del sistema. Este servidor contiene varios *daemons* que realizan las operaciones necesarias y son arrancados según la configuración establecida mediante el programa *condor_master*. Este proceso controla el resto de procesos de Condor de manera automática.

Para constituir el *pool* de ordenadores que ejecuten trabajos enviados por el *Central Manager* se dispone de ordenadores dedicados a la ejecución de trabajos. Todos los ordenadores del *pool* no son estrictamente dedicados, ya que algunos de ellos pueden ser ordenadores de usuarios que ejecutan tareas cuando el usuario no esté trabajando. En este caso, los ordenadores pertenecen al *pool* a tiempo parcial, utilizando el tiempo que el usuario no usa el ordenador para ejecutar los trabajos.

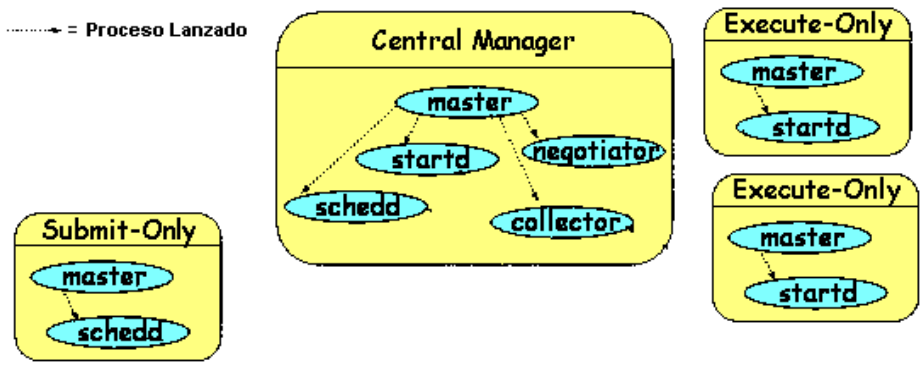

Figura 4 - Procesos Condor en los diferentes roles

Los roles que pueden presentar los ordenadores en un sistema Condor son tres:

- Gestor Central (*Central Manager*) → Su función resulta fundamental dentro del sistema, dado que es el ordenador encargado de administrar los recursos y asignar las ejecuciones de trabajos a los ordenadores dedicados a la ejecución de éstos. Periódicamente recopila información sobre el estado de todo el sistema. Tan sólo puede existir un gestor central en un mismo sistema.
- Ordenadores dedicados a la ejecución (*execute computers*) → Como su nombre indica, su tarea consiste en ejecutar los trabajos que son asignados por el gestor central. Para ejecutar los trabajos, básicamente, es necesario un programa Condor ejecutándose llamado *condor_startd*, que es el encargado de comunicarse con el *Central Manager*. Este programa realiza las funciones de envío de características del nodo y recibe los trabajos a ejecutar desde el *Central Manager*.
- Ordenadores de envío de trabajos (*submit computers*) → Son los ordenadores a los que se les otorga la capacidad de enviar trabajos al sistema. Este ordenador acostumbra a ser propiedad del usuario investigador que requiere de Condor para ejecutar su proyecto. Él es el encargado de enviar el trabajo al *Central Manager* junto con los requisitos y deseos necesarios que el trabajo necesita. El *Central Manager* realizará el *matchmaking* correspondiente entre el trabajo y los recursos publicados por los nodos del *pool* y enviará a ejecutar el trabajo en los ordenadores del *pool* que resulten los más adaptados para el trabajo. Una vez ejecutado se devolverá el resultado al investigador, ya sea mediante las llamadas a sistema remotas o a través del sistema de transferencia de ficheros que Condor implementa. El programa encargado de realizar estas tareas es el llamado *condor_schedd*. Es posible utilizar el ordenador encargado del envío de trabajos como participante en el *pool* para ejecutar trabajos.

Estos roles no son excluyentes, de manera que un solo ordenador puede actuar de gestor central, enviar trabajos y ejecutarlos.

2.2.3 - *Daemons*

Para realizar los roles definidos en la arquitectura tenemos *daemons* que realizan las siguientes funciones. Existen siete *daemons* diferentes:

- *condor_master*. Se ejecuta en todos los ordenadores del sistema. Su función es simplificar la administración del sistema. Es el encargado de lanzar la ejecución del resto de *daemons* localmente, en el propio ordenador en el que se ejecuta, y de vigilar el correcto funcionamiento de éstos. Si algún *daemon* interrumpiera por cualquier motivo su ejecución, se encargaría de relanzarlo. Además, mediante este *daemon* se pueden detener y reconfigurar el resto.
- *condor_collector*. Se ejecuta en el gestor central. Este *daemon* recoge periódicamente toda la información del estado del sistema, recibe y trata peticiones procedentes del resto de *daemons* de los demás ordenadores.
- *condor_negotiator*. Se ejecuta en el gestor central. Es el encargado de asignar trabajos a ordenadores dependiendo de los requerimientos y características tanto de los primeros como de los segundos, ambas partes han de satisfacer sus requerimientos mutuamente.
- *condor_startd*. Se ejecuta en todos los ordenadores con rol de ejecución. Este *daemon* se considera como el representante del ordenador dentro del sistema. Tiene la capacidad de iniciar, parar y suspender trabajos, y es el encargado de lanzar el *daemon starter* con la configuración apropiada dependiendo del tipo de trabajo.
- *condor_starter*. Se ejecuta en todos los ordenadores con rol de ejecución. Se encarga de monitorizar y controlar el trabajo en ejecución. Proporciona información de estado al ordenador desde el que se envió el trabajo. En ordenadores con múltiples procesadores, existe una instancia de este *daemon* por cada CPU.

- *condor_schedd*. Se ejecuta en todos los ordenadores con rol de envío. Es el encargado de gestionar la cola local de trabajos y de solicitar recursos para la ejecución de los trabajos que se encuentran parados.
- *condor_shadow*. Se ejecuta en todos los ordenadores con rol de envío, mientras el trabajo se encuentra en ejecución. Se puede considerar como el representante del trabajo en el ordenador que lo ha enviado. Existe uno por cada trabajo activo y, entre otras, tiene las funciones de transferir archivos necesarios como *logs* y estadísticas, y de efectuar llamadas remotas al sistema en caso de que sean necesarias, es decir, de la comunicación con el ordenador que ejecuta el trabajo.

El siguiente esquema muestra la configuración de un sistema Condor típico, así como la interacción entre sus *daemons*:

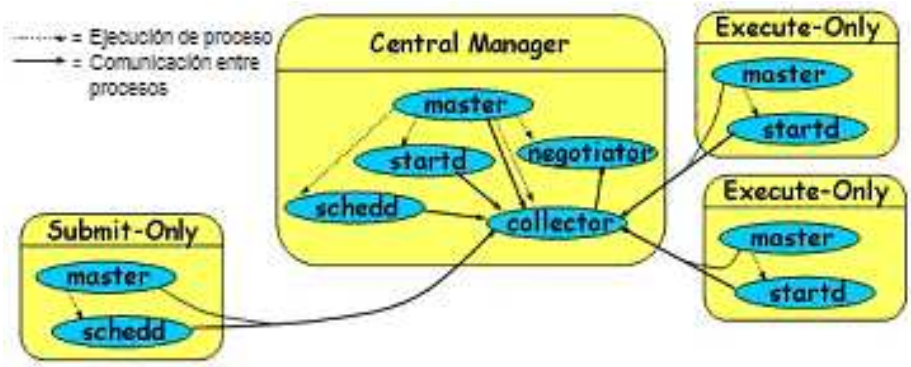

Figura 5 - Procesos Condor, interacciones entre procesos

Como se puede apreciar, los *daemons master* arrancan el resto de los procesos en cada ordenador y el *master* de todos los ordenadores que participan en el proceso se comunican con el proceso *condor_collector*, que es el encargado de recibir el estado de todo el sistema.

2.2.4 - Matchmaking con ClassAds

Es importante entender el método utilizado por Condor para acoplar recursos que pueden ejecutar trabajos según sus peticiones. Entender el *framework* que Condor utiliza para realizar este trabajo es la clave.

Dentro de las necesidades del proyecto, se pueden programar los trabajos enviados con una serie de requerimientos que el gestor intenta hacer cumplir. El investigador indica al gestor los requisitos obligatorios y los requerimientos no obligatorios que cree necesarios, como condiciones para ejecutar los trabajos, y el gestor controla los trabajos creándolos dentro de la cola (*queue*), intentando cumplir los requerimientos recibidos y enviando los trabajos a los ordenadores correspondientes que cumplen las restricciones para ejecutarlos. Por ejemplo podemos indicarle al trabajo que se ejecute en un ordenador que contenga una determinada cantidad de memoria RAM o superior. El gestor buscará entre los recursos disponibles que están en el *pool*, un recurso que cumpla las condiciones especificadas en el trabajo para ejecutarse.

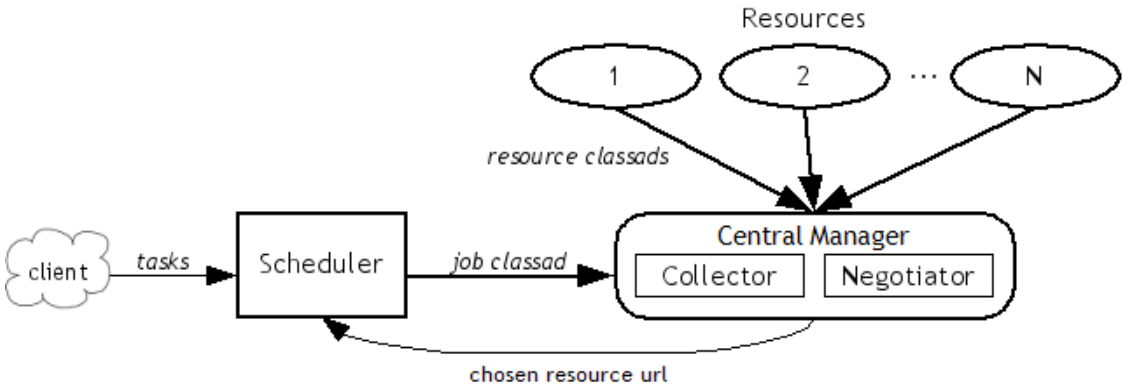

Figura 6 - Esquema representativo del mecanismo de *matchmaking*

ClassAds funciona de manera similar a los anuncios clasificados de un periódico. Todos los ordenadores en el *pool* de Condor muestran las propiedades de sus recursos en los anuncios de los recursos, tanto estáticos como dinámicos, tales

como la cantidad de RAM disponible, el tipo de CPU, la velocidad de la CPU, etcétera.
Un usuario especifica un anuncio de petición de recurso cuando envía un trabajo. La petición define tanto las propiedades requeridas como las deseadas de los recursos donde se quiere que se ejecuten los trabajos. Condor actúa como un *broker* haciendo coincidir los anuncios de los recursos ofrecidos con los anuncios de los recursos requeridos, realizando que todos los requerimientos en ambos anuncios sean satisfechos. Cuando obtiene una coincidencia envía la dirección del recurso al cliente para que conozca el recurso donde se enviará su trabajo.

2.3 - BOINC

Berkeley Open Infrastructure for Network Computing (BOINC) es un *middleware* o infraestructura no comercial para la computación distribuida usando recursos voluntarios de computación, desarrollada originalmente para el proyecto SETI@home, pero que actualmente es utilizada por diversos proyectos en campos como la física, la medicina, la biología molecular o la climatología.

Los proyectos que trabajan sobre esta infraestructura tienen un denominador común y es que requieren una gran capacidad de cálculo. La intención por la cual los proyectos utilizan la plataforma BOINC es obtener una enorme capacidad de computación utilizando ordenadores personales.
Los ordenadores que ejecutan los proyectos son casi exclusivamente ordenadores de voluntarios y los proyectos que utilizan la plataforma son en su gran mayoría sin ánimo de lucro y suelen estar dirigidos por universidades o entidades públicas.
Las tareas de los proyectos realizan cálculos complejos y BOINC divide esta información en fragmentos que envía a ejecutar en los ordenadores de los usuarios. Éstos tienen el software necesario instalado en su ordenador doméstico y ejecutan las tareas recibidas que contienen complejos cálculos para devolver, posteriormente, el resultado.

Actualmente BOINC es desarrollado por un grupo con sede en Berkeley, en la Universidad de California, está dirigido por David Anderson, director del proyecto SETI@home y recibe diversas aportaciones de desarrolladores de todo el mundo. La plataforma BOINC es considerada como un casi-superordenador, disponiendo de aproximadamente 560.000 ordenadores activos en todo el mundo y con un rendimiento medio de 1 *PFLOPS*, superando el superordenador *Blue Gene* (datos obtenidos con fecha 7/4/2008).

Esta plataforma de software está desarrollada bajo la filosofía de código abierto y protegida por la licencia GNU LGPL. BOINC se encuentra disponible para las plataformas Microsoft Windows, GNU/Linux y diversos sistemas Unix (Solaris, BSD, Mac OS X).
BOINC está diseñado como una estructura libre que permite a cualquier usuario convertirse en voluntario y participar en la ejecución de tareas de un proyecto. BOINC utiliza los ciclos que quedan libres en el procesador para procesar tareas del proyecto en el que previamente se ha registrado el usuario.

2.3.1 - Características

Las características que los sistemas BOINC nos ofrecen son:

- Recursos compartidos entre proyectos independientes → Múltiples proyectos pueden usar BOINC. Cada proyecto es independiente del resto, cada participante puede decidir los proyectos en los que desea colaborar y la cantidad de recursos a destinar en ellos. Cuando un proyecto no está accesible, los ordenadores de los voluntarios dividen los recursos de ese proyecto inactivo entre el resto de proyectos.
- Características de los proyectos → BOINC proporciona características que simplifican la creación de nuevos proyectos y las operaciones con ellos.
    - Disponibilidad de un *framework* de aplicaciones flexible. En las aplicaciones realizadas con lenguajes como C, C++ o Fortran las modificaciones a realizar son mínimas o nulas.

- Agregación de nuevas aplicaciones. El BOINC proporciona otras facilidades como añadir nuevas versiones desarrolladas de las aplicaciones al proyecto sin la acción del usuario de una manera sencilla y transparente.
- Seguridad. BOINC utiliza un sistema *Public Key Infrastructure* (PKI) de claves criptográficas. Se realizan firmas digitales de los elementos enviados, utilizando claves públicas creadas por el proyecto para firmar datos y programas.
- Múltiples servidores y tolerancia a fallos. El servidor BOINC es extremadamente efectivo, capaz de gestionar millones de trabajos por día y su arquitectura es altamente escalable. Los servidores pueden dividirse en servidores de programación y servidores de datos, pudiendo disponer de varios de ellos. Los clientes se conectan alternativamente y en el caso que no hubiera ningún servidor activo los clientes realizarían pruebas de conexión con tiempos cada vez mayores para evitar una saturación de peticiones al volver a funcionar.
- Disponibilidad del código fuente. Auque BOINC es distribuido con la licencia LGPL, las aplicaciones no están sujetas a ser código libre o seguir teniendo esta licencia.
- Soporte para gran cantidad de datos. BOINC soporta aplicaciones que producen o consumen una gran cantidad de datos. Los usuarios pueden determinar límites de uso de disco o de ancho de banda y el servidor enviará los trabajos solamente a los ordenadores que pudieran ejecutarlos.

- Características de los participantes → BOINC proporciona las siguientes características a los usuarios.
  - Múltiple plataforma. La aplicación cliente de BOINC se encuentra disponible para los sistemas operativos Mac OS X, Microsoft Windows, GNU/Linux, Sun Solaris y otros sistemas Unix.
  - Facilidad de uso. La instalación de la aplicación cliente y su uso son sencillos y fáciles de comprender. BOINC está diseñado para todo tipo de público e intenta ser atractivo de usar.

- Interfaz basada en web. BOINC dispone de interfaces web para que el usuario realice las gestiones más usuales como la creación de una cuenta, editar las preferencias, enviar mensajes privados a otros usuarios y crear comunidades *online*.
- Caché configurable. El cliente BOINC descarga suficiente trabajo para mantener el ordenador trabajando por el espacio de tiempo descrito en las preferencias del usuario. De esta forma, indirectamente, se permite configurar la cantidad de conexiones que el cliente realiza, controlando el ancho de banda usado por el cliente para descargar elementos del proyecto.
- Arquitectura extensible. BOINC proporciona interfaces documentadas de los componentes del sistema y permite que otros usuarios desarrollen elementos o aplicaciones que lo amplíen.

2.3.2 - Aplicaciones ejecutables por BOINC

BOINC ha sido diseñado para soportar aplicaciones que tengan requisitos de computación intensiva y/o requisitos de almacenamiento.
El principal requisito que debe cumplir la aplicación debe ser la divisibilidad en un gran número (miles o millones) de trabajos que puedan ejecutarse independientemente.

Si el proyecto usara recursos voluntarios, tendría unos requisitos adicionales:

- Atracción del público → La aplicación debe ser vista interesante para el público y así atraer a un gran número de participantes.
- Bajo ratio de datos/computación → La entrada y salida de datos se envían a través de conexiones a Internet, pudiendo ser caras y/o lentas. Como regla se determina que si la aplicación produce o consume más de 1 GB de datos por día, entonces es mejor y más barato ejecutar la aplicación en un *cluster* que con recursos voluntarios.
- Tolerancia a fallo → Un resultado devuelto por un voluntario no puede ser tomado como válido. Para solucionar este problema se utiliza computación

redundante que nos ayuda a reducir la probabilidad de error. Si la aplicación requiere una probabilidad de error del 0%, este sistema no es el adecuado para ejecutar el proyecto.

2.3.3 - Arquitectura

El modelo que BOINC utiliza se corresponde con un modelo cliente-servidor, donde el elemento principal es el servidor central del proyecto, el cual es el encargado de enviar trabajos (llamados *workunits*) a los ordenadores de los voluntarios que colaboran ejecutando las tareas del proyecto. Cuando el ordenador no se encuentra en uso, encontrándose entonces en estado ocioso, BOINC se activa y comienza a ejecutar las *workunits* de los proyectos a los que el usuario está suscrito.

Las funciones que realiza el servidor se pueden separar en dos tipos de servidores:

- El primer tipo de servidor es un servidor que se encarga de la distribución de tareas y de la comunicación con los clientes.
- El segundo tipo de servidor es un servidor que puede contener las bases de datos de todo el proyecto.

Estos servidores no son exclusivos, pueden encontrarse ambos a la vez y pueden coexistir múltiples servidores que realizan las mismas funciones, posibilitando una arquitectura basada en múltiples servidores que distribuyan las tareas y diversos servidores que realicen el almacenamiento de datos. Así conseguimos una redundancia a nivel de servidor que nos permite que el proyecto no esté disponible por el fallo de un servidor.

Los clientes se conectan alternativamente a los diferentes servidores distribuidos, comportándose todos ellos transparentemente al usuario y dando la sensación de unicidad.

Los clientes por su parte, son usuarios que deciden participar voluntariamente en los proyectos que habitualmente están abiertos a cualquier usuario que decida participar.
Solamente es necesario registrarse en los proyectos mediante la aplicación cliente de BOINC o en la página web del proyecto introduciendo un correo electrónico y una contraseña elegida para acceder a sus estadísticas de progreso. Es posible que los proyectos, debido a un superávit de usuarios o por mantenimiento, no admitan usuarios, pero suele suceder de forma temporal.
A través de la interfaz web de los proyectos es posible inscribirse grupos de usuarios, personalizar la cuenta para determinar el tiempo de funcionamiento o el espacio de disco duro del que se puede disponer en el equipo voluntario.

Una vez los usuarios han instalado el cliente BOINC y se han registrado al proyecto, el ordenador del usuario se conecta al servidor y realiza una petición para descargar *workunits* a ejecutar.

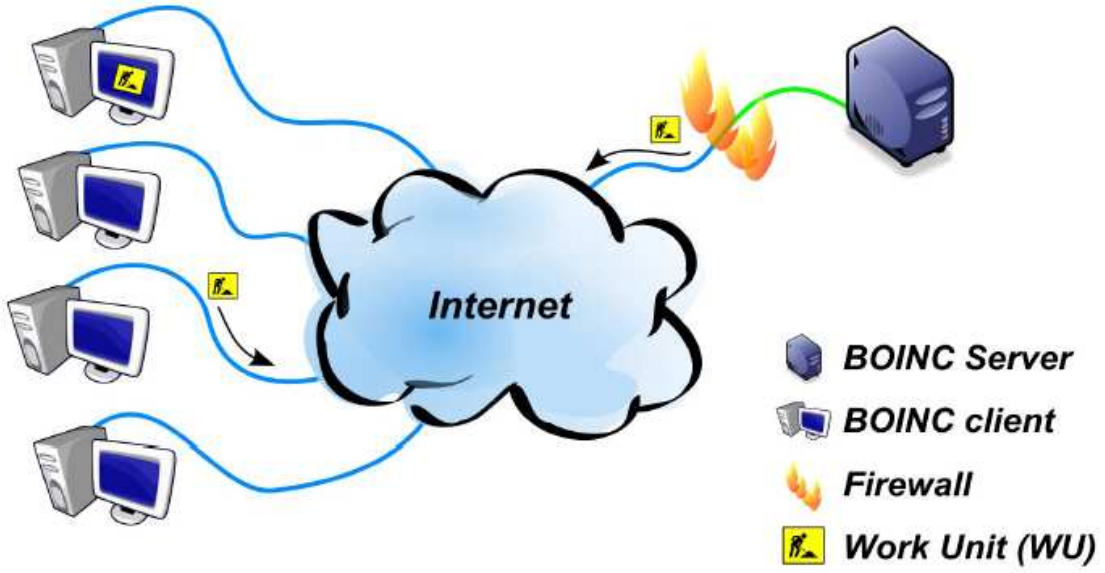

Figura 7 - Arquitectura del sistema BOINC

El programa cliente usa características de configuración como la cantidad de disco que se permitirá que el proyecto utilice como almacenamiento, lo que repercute directamente en la cantidad de descargas que se pueden realizar. A mayor cantidad de disco disponible se podrá descargar un mayor número de *workunits* de una vez y almacenarlas en el disco del ordenador voluntario.
Como vemos en la figura siguiente, un ordenador pide una *workunit* y devuelve los resultados antes de pedir una nueva *workunit*, mientas que el segundo ordenador realiza una petición de 3 *workunits* y devuelve los resultados de cada *workunit* a medida que va disponiendo de ellos.

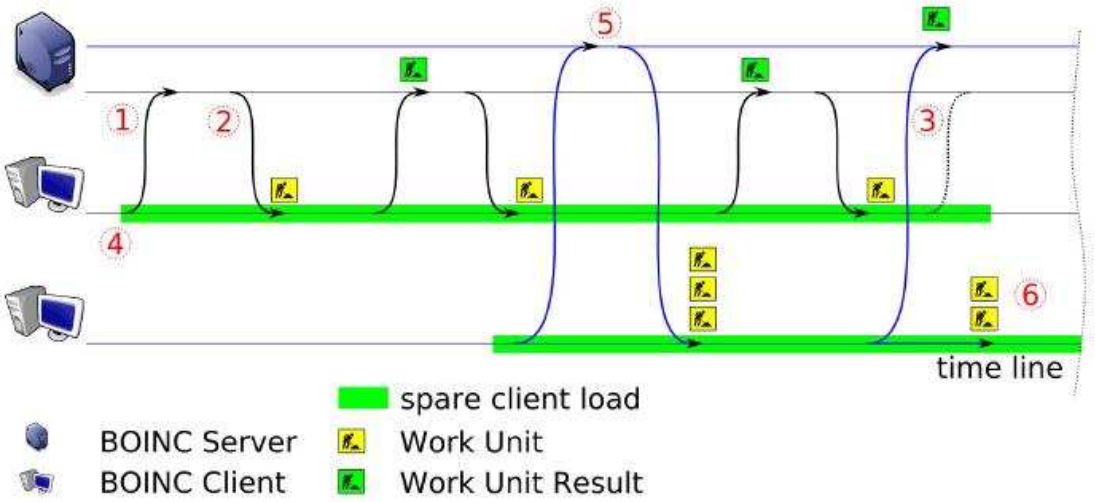

Figura 8 - Diagrama de tiempos según la configuración del espacio en disco

El servidor de BOINC se compone de diferentes programas para realizar las funciones. Como parte de BOINC, destacamos dos Interfaces de Entrada Común (*Common Gateway Interface,* CGI) que manejan las peticiones de los clientes, enviando nuevas tareas y recibiendo los resultados.

Todo servidor BOINC también dispone de, como mínimo, 3 *daemons* que realizan el trabajo. Estos *daemons* son llamados *feeder*, *file_deleter* y *transitioner*.

El primero, *feeder*, es el encargado de extraer *workunits* de la base de datos y rellenar *slots* de tiempo que el *scheduler* lee para enviar a los clientes voluntarios.
El segundo de ellos, *file_deleter*, tiene la función de borrar aquellos ficheros que no se volverán a utilizar en el servidor, ya sea porque se han procesado o porque son ficheros de entrada ya enviados y no van a ser necesitados más.
El tercer *daemon* necesario es el *transitioner*, que se encarga de gestionar las transiciones en el servidor entre estados de las *workunits* y los resultados.

Otros *daemons* utilizados, pero no estrictamente necesarios son *validator* y *assimilator*.
El primero, como su nombre indica, valida los resultados. Si los resultados cumplen las normas establecidas en la validación implantada por el administrador del proyecto se asignan créditos a los usuarios.
El *assimilator* por su lado procesa los resultados validados. Puede actuar guardando los resultados en una base de datos o generando nuevas *workunits* a partir de los resultados obtenidos y validados usando código específico del proyecto.

El cliente por su parte, requiere de forma obligatoria dos *daemons* o programas ejecutados por un sistema de programación de tareas como *cron*. Estos programas son *boinc* (o *core_client*) y *boincmgr*.

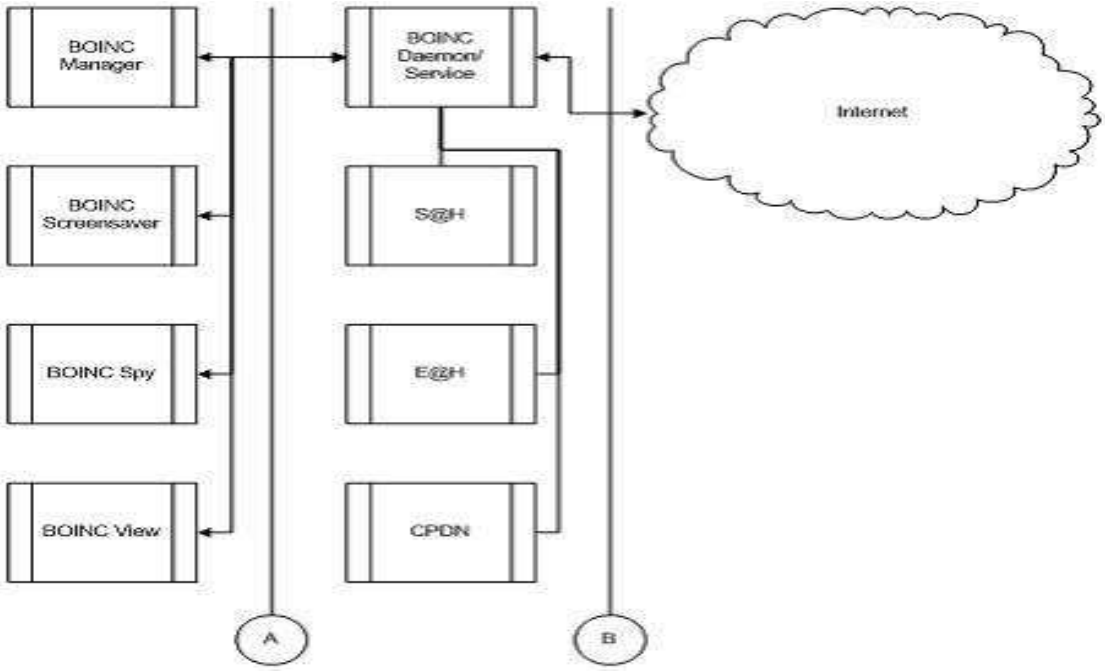

Figura 9 - Estructura de la aplicación cliente de BOINC

El primero de ellos, *core_client*, es el núcleo del cliente y gestiona las comunicaciones con el servidor, la recepción de *workunits* y envío de resultados, así como la actualización de los proyectos.
El segundo, *boincmgr*, es una interfaz gráfica programada en WxWidgets que nos permite interactuar con el *core_client* de una forma más sencilla.
Pueden existir otros componentes más en el cliente como el salvapantallas utilizado cuando BOINC se ejecuta o programas que consultan las estadísticas del trabajo realizado en cada proyecto registrado. Estos procesos son manejados por el *daemon boincmgr*.

Los componentes en el cliente se comunican mediante Llamadas a Procedimientos Remotos (*Remote Procedure Calls*, RPC) y suelen estar ubicados en un mismo ordenador, aunque se puede configurar el *core_client* para ser gestionado de forma remota mediante un usuario y contraseña.
En cambio el cliente core_client se comunica con las aplicaciones y los elementos de los proyectos científicos mediante memoria compartida.

2.3.4 - Sistema de créditos

El sistema de créditos de BOINC proporciona varias funcionalidades añadidas al proyecto. Los créditos no solamente nos indican cuanto trabajo hemos realizado hasta ahora, si no que también nos indica si un ordenador es apto para recibir un trabajo.

El sistema de créditos de BOINC está diseñado para evitar que los usuarios pueda hacer trampas cuando envían un resultado que no es correcto. Antes de asignar los créditos a los usuarios se tiene que revisar los resultados recibidos en la validación de resultados. Las razones que implican tener la necesidad de un sistema fiable de créditos son las siguientes:

- Asegura que los resultados obtenidos sean científica y estadísticamente válidos. Es necesaria una fuente estadística fiable para comprobar los resultados y se extrae de las estadísticas realizadas a los usuarios según sus créditos.
- Para preparar y repartir los envíos se hacen modelos de complejas variables que nos ayudan a determinar a quien enviar los trabajos. Los ordenadores capacitados para devolver un resultado correcto serán aquellos que hayan recibido una *workunit* y hayan devuelto un resultado correcto.
- No hay razón específica por la que una persona quiera donar sus ciclos de CPU. Se tiene que controlar el mal uso del sistema, ya que el anonimato que nos proporciona el sistema es un riesgo que podría acabar con el proyecto.

Los créditos se miden en *cobblestones*, llamado así por Jeff Cobb, desarrollador del proyecto SETI@home.

El sistema se basa en el sencillo concepto de que 100 *cobblestones* equivale a un día de trabajo en un ordenador voluntario que tenga 1,000 MIPS basado en el *benchmark de Whetstone* o 1,000 MIPS basado en el *benchmark de Dhrystone*.

El único punto desfavorable de este sistema es que se necesita resolver la unidad de trabajo para saber cuantos créditos se obtienen por ello. Al enviar la unidad de trabajo resuelta, se piden una cantidad de créditos determinados, que dependiendo de los cálculos realizados podrán ser los que finalmente se concedan, más de los que se concedan o menos.

Para realizar el cálculo de créditos, BOINC usa *Benchmarks,* programas que miden la velocidad de un sistema. Calculando el tiempo usado para resolver la *workunit* otorga un número de créditos determinado.
Actualmente se usa más de una variable para ese cálculo, como la cantidad de memoria RAM o la velocidad de la CPU.
Las discrepancias entre los créditos otorgados y merecidos suelen ser grandes.

La mayoría de los proyectos han llegado a un consenso para entregar un número determinado de créditos por cada misma unidad de trabajo.
Los créditos se suman en la cuenta que el usuario dispone en el mismo proyecto.

Cuando un ordenador recibe una unidad de trabajo y la devuelve resuelta no obtiene créditos inmediatamente, sino que se pide una determinada cantidad de créditos.
Luego cada proyecto valida los datos obtenidos mediante el *daemon validator* que se haya programado para cada proyecto.
Una vez validados, se conceden los créditos que se cree que merece la ejecución de la *workunit*, que pueden ser más, menos o igual a los pedidos.
Si la unidad de trabajo se entrega resuelta más tarde del plazo previsto o si la comprobación no valida el resultado, no se le asignarán créditos.

Se puede comprobar el crédito en la sección *Your credit* en la página web de la cuenta del usuario para cada proyecto.
Los proyectos BOINC exportan la información estadística de los créditos acumulados en forma de archivos XML que pueden ser descargados por ordenadores en todo el mundo. Muchas páginas web han desarrollado sistemas para mostrar estas estadísticas y muchas de éstas las muestran en forma de gráfica que puede plasmarse en páginas personales.

Como BOINC permite la creación de grupos también se permite visualizar los créditos que cada grupo acumula.

Capítulo 3 - Arquitectura y requisitos

En una primera fase del proyecto se estudiaron diferentes problemas que eran necesarios tratar. Se estudió la viabilidad a todos ellos y se explicarán las propuestas con sus pros y contras, y que se ha decidido realizar para solucionar cada una de ellas. También se revisarán los principales problemas que han surgido durante la implementación, la revisión de los recursos utilizados, las tareas realizadas y las soluciones propuestas.

Se comenzará explicando la arquitectura del sistema, las ideas iniciales y sus características. Las arquitecturas de BOINC y Condor son básicas, ya que la arquitectura final es la combinación de las dos. La arquitectura definitiva es estudiada y explicada en este punto.
Las características y los problemas serán los que más tiempo nos supondrán, pues su estudio y métodos de resolución son los más importantes.

3.1 - Arquitectura del sistema

La arquitectura adoptada por el proyecto es la mezcla de las dos arquitecturas anteriormente vistas, Condor y BOINC. Las arquitecturas de los sistemas no se han modificado, pero se han juntado para obtener la unión entre ambos sistemas.

Las arquitecturas anteriormente vistas nos proporcionan sus propios sistemas para ejecutar trabajos y gestionar los recursos que tienen.
Por un lado se dispone de un *pool* de Condor que permite ejecutar trabajos especificados con *ClassAds* dentro de la red local y que gestiona los ordenadores que se encuentran dentro del *pool* obteniendo información de ellos para conocer que trabajo se les puede enviar a ejecutar, siendo éste apto para el recurso.
Por otro lado se dispondrá de los ordenadores voluntarios conectados mediante el sistema de computación voluntaria BOINC ejecutarán las *workunits* que el servidor entregue e irán devolviendo los resultados a medida que los obtengan. El servidor de BOINC desconoce el número de recursos que están ejecutando los procesos en ese momento, pero dispone de sistemas para que los trabajos siempre tengan un resultado posible.

Desde el proyecto se quiere unir los dos sistemas obteniendo la fiabilidad y control de Condor sobre los trabajos y los recursos y la facilidad de uso y la disponibilidad de una gran cantidad de ordenadores que BOINC proporciona.
Para realizar esto se ha querido crear una *workunit* de BOINC que represente un sistema Condor que se pueda unir al *pool* sin problemas, de forma transparente y segura.
En este caso, estas *workunits* a ejecutar son procesos de Condor que permiten introducir el ordenador voluntario en el *pool* y recibir aquellos trabajos Condor que se adapten a las características requeridas por el trabajo y por el ordenador voluntario.

El funcionamiento de la arquitectura es la unión de ambos sistemas. Previamente el usuario debería instalar el cliente BOINC en su ordenador. Dependiendo del sistema

operativo usado existen diferentes formas de instalación, pero este es un proceso sencillo.

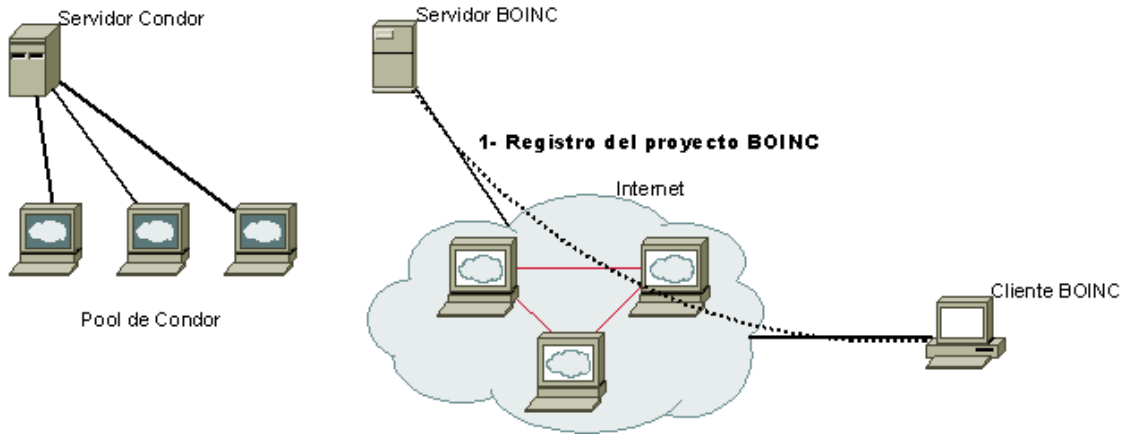

Figura 10- Conexión con el servidor BOINC

Una vez instalado, en la primera ejecución del cliente, éste pedirá realizar el registro en el proyecto que se desea colaborar.

Cuando el cliente nos pide el identificador del proyecto se refiere a la dirección URL del mismo. Los proyectos disponen de su propia página web que se construye al crear el proyecto en el servidor, siendo parte de la interfaz web que el proyecto BOINC proporciona. Esta página web se utiliza como identificador de cada proyecto.

Ya registrado, el cliente conectará con el servidor. Éste enviará una *workunit* a ejecutar para satisfacer la petición recibida. En esta *workunit* se indican los ejecutables y ficheros de datos que son necesarios para la ejecución de la tarea. En nuestro caso serán los ficheros necesarios para arrancar Condor en el ordenador voluntario configurado correctamente para anexarse en el pool.

El cliente esperará que el ordenador se encuentre inactivo para iniciar la ejecución del proyecto.

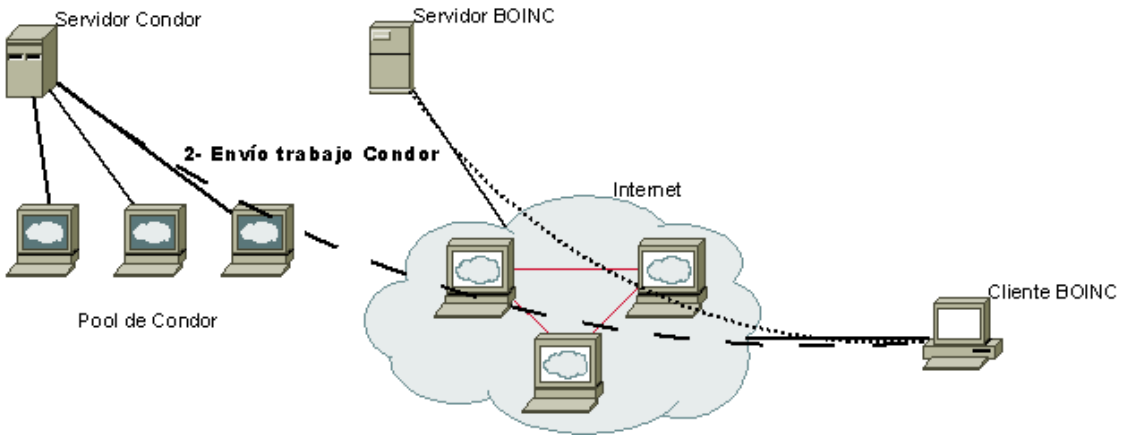

Figura 11 - Conexión con el servidor Condor

Mientras el ordenador permanece inactivo de los procesos del usuario, el proyecto BOINC iniciará su ejecución y Condor arrancará configurado oportunamente para conectarse al *pool*, publicando su estado y sus recursos mediante los *ClassAds* y el *Central Manager* podrá encontrar oportuno enviar algún trabajo a ejecutar en el ordenador voluntario.

Si el trabajo que Condor envió al ordenador se ejecutara correctamente, el resultado sería enviado al creador del trabajo Condor, como si fuera un trabajo que hubiera sido ejecutado en cualquier otro ordenador del *pool*.
Si BOINC finalizara la ejecución realizaría una validación del resultado obtenido por la *workunit* enviada y en caso de detectar que el resultado es correcto sumaría nuevos créditos a la cuenta del usuario.

Si el trabajo no pudiera finalizar porque el usuario volviera a utilizar el ordenador y la aplicación fue enlazada con las librerías de Condor, Condor podría realizar un *checkpoint* y migrar el proceso para continuarlo en otro ordenador. Si no fuera así, el trabajo finalizaría juntamente con la ejecución de Condor y BOINC.

Condor, por su parte, dispone de sistemas para evitar que los trabajos se pierdan o generen un error si se diera el caso anterior y no se pudiera realizar la migración a otro ordenador. La fiabilidad de Condor nos asegura un resultado si los recursos los pueden dar.

Cuando el ordenador volviera a su estado de inactividad,. BOINC volvería a ejecutar la *workunit* que ya tuviera descargada en el ordenador voluntario, iniciando de nuevo Condor y estando preparado para ejecutar cualquier trabajo que el *Central Manager* de Condor considerara oportuno de volver a enviar y ejecutar.

3.2 - Implementación

La implementación que se ha realizado en el proyecto se ha escogido después de haber sido estudiada y haber descartado otras por no cumplir con los objetivos y necesidades marcados en el proyecto, vistos en capítulos anteriores. Todas las soluciones que se verán solucionan los problemas surgidos y estudiados.

3.2.1 - Facilidad de instalación

La instalación de Condor es un proceso que se ha simplificado al ejecutar un programa que viene incorporado en la versión cuando se descarga de la página web del proyecto. Este programa realiza preguntas sobre el tipo de sistema o rol que deseas instalar en ese ordenador, el usuario que realizará la instalación, el uso o no de sistemas de ficheros compartidos por red o la ubicación del ejecutable de la máquina virtual de Java.

Este proceso se puede repetir varias veces para corregir cualquier error que hayamos cometido.

Es un sistema sencillo y rápido de realizar la instalación para cualquier tipo de ordenador dentro de la arquitectura de Condor, pero después de la instalación se necesita igualmente modificar los ficheros de configuración de Condor, *etc/condor_configure* y *condor_configure.local*, situados en el directorio correspondiente de la instalación de Condor.

La modificación obligatoria que Condor obliga a hacer en el primer archivo visto es un valor que indica los ordenadores a los que se le permite realizar conexiones. Este valor por defecto está comentado para que no se pueda permitir a ningún ordenador.

Como se puede apreciar, la instalación de Condor no es tan automática, pero se necesita conocer los cambios a realizar para que el sistema funcione correctamente.

La instalación de BOINC es un proceso bastante diferente. Cada sistema operativo dispone de un método de instalación diferente. En los sistemas Linux, que ha sido el sistema utilizado, cada distribución utiliza su propio método para realizar la instalación de software. En los sistemas Windows deberemos ir a la página web oficial de BOINC y descargar un ejecutable que nos instalará el programa.

Siguiendo este proceso el cliente BOINC estará instalado y esperando poder registrarse en un proyecto para colaborar.

El servidor de BOINC, por otro lado, es difícil de configurar, ya que dispone de diversas herramientas como Apache o MySQL que requieren de una configuración adecuada.
Una vez el servidor es configurado, la realización de un proyecto es un proceso sencillo de seguir. Al proyecto se le añadirán las aplicaciones y se crearán las *workunits* a ejecutar.

3.2.2 - Seguridad

La primera necesidad que nos surge es la obligatoriedad de disponer de un método seguro para proteger ambos entornos. Estos sistemas deben asegurar que el usuario es un usuario válido y que sus acciones no pueden comprometer los sistemas.

En el caso de BOINC no es sencillo realizar esta acción ya que una de las particularidades del sistema es el anonimato de los usuarios. No se puede confiar en los usuarios que ejecutan los trabajos desde el punto de vista de BOINC, pero en

Condor todos los usuarios son conocidos y se confía en ellos al enviar trabajos o al ejecutarlos.
Igualmente los dos sistemas incorporan métodos para prevenir posibles fallos de seguridad.

BOINC no dispone de un sistema que certifique las *workunits* enviadas. Éstas se encuentran almacenadas en el servidor y es él el encargado de distribuirlas. Se confía en el desarrollador de las *workunits* y en su buena fe como usuario propietario del proyecto que desea que se lleve a buen término.
En cambio BOINC realiza firmas criptográficas de los ficheros enviados a ejecutar en las *workunits* mediante un sistema de clave pública PKI, realizando un sistema de comprobación similar a las funciones *hash* utilizadas para realizar comprobaciones de ficheros descargados en Internet. Cualquier modificación de un fichero el servidor lo detectará y no lo dará como válido dentro del sistema.
Este sistema nos asegura que los ficheros y sus resultados no están comprometidos.

Condor también aporta sus sistemas de seguridad. Aunque se encuentra en un sistema controlado puede ocurrir que un usuario quisiera eliminar o modificar algún trabajo que se encontrara en la cola. Para ello Condor almacena la configuración de los procesos *condor_schedd* que enviaron el trabajo, no permitiendo la modificación en caso de no corresponderse. Condor está diseñado para ejecutarse con permisos de *root* en los ordenadores donde se ejecuta, pero muchas veces no puede darse esta circunstancia. En ese caso, cuando Condor se ejecuta con permisos de un usuario, necesita de directorios donde pueda crear y borrar los ficheros que utiliza.

3.2.3 - Limitación de usuarios

Una de las mayores ventajas de BOINC es que su escalabilidad es tremendamente dinámica. Esta escalabilidad abarca una gestión autónoma de los usuarios o nodos, lo que a su vez reduce las tareas administrativas. La configuración del proyecto por defecto tiene deshabilitada la creación de usuarios por medio del interfaz web. Para habilitar esta característica, hay que modificar el fichero *config.xml*, que se encuentra

en el directorio raíz del proyecto, cambiando la entrada <*disable_account_creation*> de 1 a 0.

Otra sistema de realizar la creación de usuario es a través del mismo cliente BOINC. Cuando un usuario se registra en el proyecto con su dirección de correo electrónico y su contraseña está dando de alta el usuario dentro del servidor.

Igualmente se debe configurar en el servidor la activación de creación de cuentas para que permita añadir nuevos usuarios.

Este sencillo sistema permite desactivar la creación de cuentas o activarla cuando sea necesario. Si el proyecto se encontrara en fase de mantenimiento o se quisiera prevenir la saturación de los sistemas si se añadieran más usuarios, se podría desactivar la creación de cuentas.

3.2.4 - Aplicación BOINC

Con la infraestructura BOINC implementada únicamente se necesitaría de una aplicación que fuera nuestro enlace con Condor. BOINC ofrece una *Application Program Interface* (API) que permite construir aplicaciones que se pueden incorporar en un proyecto, el cual los usuarios ejecuten. Esta API incorpora funciones específicas de BOINC para el control de la aplicación, realizar transferencias de ficheros o para la creación de salvapantallas, por ejemplo.

BOINC incorpora un sistema para poder ejecutar aplicaciones sin necesidad de modificarlas e incluir llamadas a la API de BOINC. Esto permite ejecutar aplicaciones de las que no disponemos su código fuente y por lo tanto no podemos modificarlas. El método utilizado es llamado *wrapper* y ejecuta la aplicación que se desea sin necesidad de realizar llamadas a BOINC, ya que es el mismo *wrapper* quien las realiza.

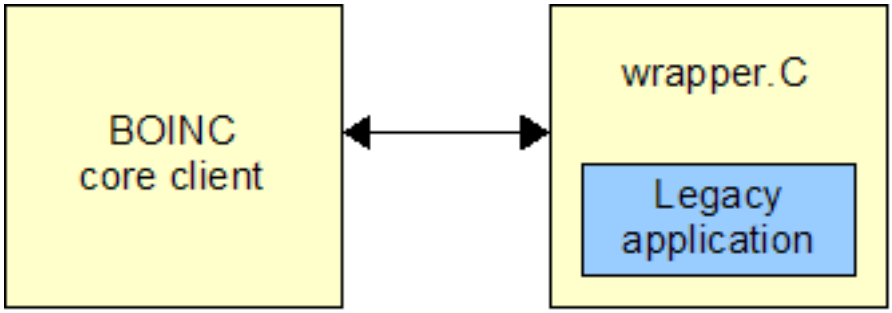

Figura 12 - BOINC *wrapper*

Ante la eventualidad de crear una nueva aplicación incluyendo funciones de la API de BOINC desde el inicio, con los consiguientes problemas que pudiera producir, se ha decidido utilizar el método *wrapper* que BOINC suministra para poder arrancar los programas de Condor necesarios.

Se ha creado en el proyecto una capa intermedia que proporciona la configuración de los elementos que Condor requiere. Esta capa intermedia es llamada *condor_wrapper* y es la tarea que el *wrapper* de BOINC ejecuta.
Este proceso adecua el entorno en el ordenador voluntario para poder ejecutar Condor con la configuración correspondiente.

3.2.5 - Workunits

Las *workunits* o unidades de trabajo son las tareas que BOINC envía a ejecutar en los ordenadores voluntarios. La infraestructura cliente descarga estas *workunits* y ejecuta las tareas que en ella se incluyen.

Una *workunit* define la aplicación y el conjunto de datos que tienen que ser ejecutados y procesados por el cliente. Las unidades de trabajo están descritas por una plantilla de unidades de trabajo y por una plantilla de resultado.
La plantilla de unidad de trabajo describe la referencia del conjunto de datos de entrada en el nodo de destino.
La plantilla de resultado, por otro lado, describe la referencia del conjunto de datos resultante. Ambas pueden crearse en el directorio */templates* del proyecto.

Cada unidad de trabajo se identifica por una ID única, que es gestionada por el servidor BOINC y la base de datos. La herramienta *create_work* se usa para pasar los trabajos del proyecto a la base de datos. A partir de aquí la *workunit* está preparada para ser descargada y ejecutada.

Las *workunits* incluyen fragmentos de trabajos que se desean ejecutar. Estos fragmentos son parte del proyecto y son los procesos que realizan una gran cantidad de cálculos computacionales.

El problema que nos encontramos fue la manera de incluir la inicialización de los procesos Condor dentro de las *workunits*. La creación de una sola *workunit* que incluyera los procesos Condor a ejecutar ya contemplaba las opciones a realizar.

Como ya se sabe, debido al sistema que BOINC implementa los ordenadores no son dedicados y entran y salen de la ejecución de tareas constantemente. Muchas ocasiones sucede que el ordenador recibe una *workunit* a ejecutar pero ese ordenador falla o se apaga y no vuelve a encenderse durante mucho tiempo o nunca más.

Para evitar esta desaparición espontánea del recurso y controlar la tolerancia a fallos de BOINC se realiza una computación recurrente. El proceso es duplicar las *workunits* que se envían para asegurar el retorno de un resultado como mínimo. Este proceso también ayuda a la hora de determinar los créditos a otorgar a cada usuario comprobando si los resultados de la misma *workunit* son iguales o parecidos.
Los parámetros que indican la cantidad de *workunits* a duplicar y los resultados mínimos para asegurar un resultado correcto los establece el administrador del proyecto en su configuración.

Condor controla la ejecución de cada trabajo. El sistema de control provoca que si un proceso no puede migrar cuando un fallo ocurre, Condor desestima el proceso y lo vuelve a enviar a otro ordenador del *pool* para que realice de nuevo la ejecución.

De esta manera se asegura la ejecución de los trabajos enviados a los ordenadores voluntarios que pueden ejecutar procesos de Condor.

Capítulo 4 - Implementación y experimentación

En capítulos anteriores se han descrito los objetivos, los sistemas utilizados y la sinopsis general del proyecto. Se han estudiado los problemas que han surgido durante la fase de análisis y la realización del proyecto. Desde la descarga del primer sistema a instalar hasta la última prueba realizada se han realizado diversos trabajos que se intentarán sintetizar. Este capítulo se centrará en conocer un poco más a fondo el trabajo realizado, descendiendo hasta el código fuente en algunas ocasiones.

Se comenzará explicando la instalación del *testbed* que nos ha servido para realizar el proyecto.
Después estudiaremos la estructura de la capa intermedia, *condor_wrapper*, que se desarrolló y es el eje central de la solución encontrada.
La experimentación realizada nos confirmará el buen funcionamiento del proyecto.

4.1 - Instalación

La instalación de los sistemas de prueba fue realizada en dos servidores independientes que ofrecen los servicios necesarios para llevar a cabo los experimentos necesarios.

Por un lado se encuentra el servidor Condor, que es uno de los sistemas básicos y necesarios dentro del proyecto. El servidor Condor es el encargado de añadir el ordenador voluntario al *pool* y poder visionarlo como un ordenador más donde poder enviar tareas a ejecutar.

Por otro lado encontramos el servidor BOINC, que es el responsable de generar la tarea primordial para que los clientes puedan colaborar dentro del entorno Condor creado.

Los dos sistemas están desarrollados para varias plataformas. En nuestro caso hemos elegido implementarlos bajo el sistema operativo GNU/Linux.
El sistema GNU/Linux es un sistema completo y maduro que nos proporciona herramientas para crear proyectos y una interfaz adecuada para el desarrollo del proyecto.
Entre todas las distribuciones del sistema operativo GNU/Linux se ha escogido Ubuntu Linux. Esta es una distribución, orientada al escritorio, aunque ofrece una versión dedicada y preparada para ejecutarse como servidor.
Ubuntu surgió de una de las más veteranas y más grandes distribuciones de software, Debian GNU/Linux. Ubuntu ha portado una cantidad de paquetes Debian a su sistema y se ha trabajado para asegurar el buen funcionamiento de todos ellos.
Debian por su parte dispone de más de 9000 paquetes de software y Ubuntu puede instalarlos en caso de necesitarlos.

La decisión de utilizar este sistema operativo, y en concreto la distribución, es la fiabilidad que nos proporciona y la compatibilidad con los sistemas utilizados para el desarrollo del proyecto.

4.1.1 - Condor

Teniendo los ordenadores con el sistema operativo instalado se procedió a descargar e instalar el sistema Condor para instalar el servidor.

Condor proporciona dos *scripts* de instalación. En nuestro caso utilizamos el script *condor_install* que nos irá haciendo preguntas para conocer la configuración que deseamos realizar. Las preguntas a realizar piden información del siguiente tipo:

1. ¿Qué ordenador será el *Central Manager*?
2. ¿A qué ordenadores se debería permitir lanzar trabajos?
3. ¿Se ejecutará Condor como *root*?
4. ¿Quién administrará los ordenadores con Condor en el *pool*?
5. ¿Se dispone de una cuenta de usuario llamada condor en todos los ordenadores y está su directorio raíz compartido?
6. ¿Dónde se debería instalar los directorios locales?
7. ¿Dónde se debería instalar los ficheros de configuración, los ejecutables, y el resto de los archivos de Condor?
8. ¿Se usa AFS?
9. ¿Se dispone de suficiente espacio de disco?

Conociendo la respuesta a estas preguntas, la instalación de Condor se realiza sin problemas.
Este proceso se puede lanzar varias veces para reconfigurar Condor o para realizar una instalación que comparte diferentes tipos de roles.

Una vez instalado el sistema, procederemos a configurarlos debidamente, pues la configuración por defecto no es satisfactoria.

El fichero de configuración se encuentra en el directorio donde hemos instalado Condor, <condor_dir>/etc/condor_configure. Este es el fichero principal y puede

existir un fichero secundario que sobrescriba la información del primero para esta instalación local. El fichero se encuentra en <condor_dir>/condor_configure.local.
Las modificaciones las podemos realizar tanto en el primer fichero como en el segundo, si éste existiera.

Dentro del fichero *condor_configure* modificaremos la opción que da nombre a nuestro *pool*.

```
##  This macro is used to specify a short description of your pool.
##  It should be about 20 characters long. For example, the name of
##  the UW-Madison Computer Science Condor Pool is ``UW-Madison CS''.
#COLLECTOR_NAME         = My Pool
COLLECTOR_NAME          = CONDOR-BOINC TEST
```

En este caso llamaremos a nuestro *pool* con el nombre CONDOR-BOINC TEST.
Éste nos servirá para diferenciarnos de otros posibles *pools* y los ordenadores que se conecten al *Central Manager* indicarán el nombre para saber a qué *pool* conectarse.

La modificación que Condor obliga a hacer es un valor que indica los ordenadores a los que se le permite el acceso de escritura sobre el ordenador. Este valor por defecto está comentado para que no se pueda permitir a ningún ordenador. Se debe permitir a los trabajos poder escribir sobre los directorios determinados que necesiten. Para ello configuraremos la siguiente opción:

```
##  Write access.  Machines listed here can join your pool, submit
##  jobs, etc.  Note: Any machine which has WRITE access must
##  also be granted READ access.  Granting WRITE access below does
##  not also automatically grant READ access; you must change
##  HOSTALLOW_READ above as well.
##
##  You must set this to something else before Condor will run.
##  This most simple option is:
##    HOSTALLOW_WRITE = *
##  but note that this will allow anyone to submit jobs or add
##  machines to your pool and is serious security risk.
#HOSTALLOW_WRITE  = YOU_MUST_CHANGE_THIS_INVALID_CONDOR_CONFIGURATION
_VALUE
HOSTALLOW_WRITE   = *
#HOSTALLOW_WRITE  = *.your.domain, your-friend's-machine.other.domain
#HOSTDENY_WRITE   = bad-machine.your.domain
```

Indicando un asterisco permitiremos que cualquier ordenador pueda escribir en los directorios que necesiten para enviar los trabajos.

La siguiente modificación es necesaria para evitar que Condor solamente trabaje cuando no hay actividad en el sistema, obligando que siempre que se le requiera ejecute los trabajos.

```
##  When is this machine willing to start a job?
#START            = $(UWCS_START)
START             = TRUE
```

Otro cambio a realizar en la configuración es la forma de indicarle a Condor donde puede encontrar los ficheros de configuración. Condor dispone de tres métodos: una variable de entorno indica la ruta (*$CONDOR_CONFIGURE*), un fichero que almacena la ruta llamado */etc/condor_configure* o situar este fichero en el directorio *~/condor/condor_configure*.

Condor por defecto no instala un arranque automático de sus *daemons,* con lo que nos obliga a realizar nuevas modificaciones para que *éstos* puedan arrancar automáticamente al encender el ordenador.

El cliente de Condor utiliza el mismo sistema de instalación y tiene las mismas configuraciones, pero hay que especificarle el rol a realizar en el momento de la instalación.

4.1.2 - BOINC

Se disponen de diferentes formas de obtener el servidor de BOINC, dentro de los repositorios de software encontramos el paquete BOINC que nos permite instalarlo juntamente con todas sus dependencias de aplicaciones necesarias para su funcionamiento, descargamos el código fuente desde su propio sistema *CVS* y lo compilamos en nuestro ordenador generando una instalación a medida o se puede utilizar una máquina virtual que han realizado desarrolladores para poder ejecutar pruebas.

En nuestro caso se ha optado por utilizar la máquina virtual por no disponer de ordenadores donde ubicar el servidor. Esta máquina virtual no está disponible para entornos reales, pero si para hacerla servir como parte de un *testbed*.

Para crear un proyecto ejecutaremos la orden siguiente, que nos construirá automáticamente la estructura de un proyecto. Simplemente con esta orden se tendrá la estructura de un proyecto:

```
$ ./tools/make_project -delete_prev_inst -user_name <username> \
-drop_db_first_project_root $HOME/projects/<project_name> \
-key_dir $HOME/projects//<project_name>_keys \
-url_base http:// /<url_base_project>/ \
-db_user <username> test_setup
```

Después de crearse el proyecto, la consola mostrará los mensajes necesarios para actualizar la configuración de Apache, incluyendo esas líneas en el fichero de configuración. Esta información se utiliza para acceder al proyecto por medio de un navegador web y los clientes BOINC.

Ahora ya se encuentra disponible el proyecto, pudiéndose acceder a él con un navegador web. La URL está definida por medio de un alias de Apache, por ejemplo, *http://mi_servidor.es/ test_setup*. Si el servidor fallara, habría que revisar el comando *chmod*, las rutas y la configuración de Apache; por otro lado, la configuración por defecto debería funcionar sin problemas.

Debemos recordar que por defecto la configuración del proyecto tiene deshabilitada la creación de usuarios. Para habilitar esta característica hay que modificar el fichero *config.xml* que se encuentra en el directorio *projects/<nombre_proyecto>*, cambiando la entrada *<disable_account_creation* de 1 a 0. De esta manera los usuarios podrán registrarse tanto vía página web como por la aplicación cliente.

Para añadir una aplicación y para ser ejecutada por los nodos, se incluye un directorio *app* donde se configuran las aplicaciones. El nombre del ejecutable hay que acompañarlo de una secuencia para que el servidor conozca a que plataforma y a que versión es:

```
<nombre_aplicación>_VERSMAYOR.VERSMENOR_PLATAFORMA
```

La versión indica que cliente BOINC puede ejecutarla y la plataforma de destino que debe tener el cliente para ejecutarla.

Lo siguiente que tenemos que hacer es modificar el fichero *project.xml* en el directorio raíz del proyecto para indicarle las plataformas donde se puede ejecutar. A continuación se ejecutan los comandos *bin/xadd* y *bin/update_versions* para que el proyecto añada la aplicación y las actualice en su versión y plataforma.

Para terminar de configurar el proyecto incluiremos *workunits* para procesar y los ficheros de resultados esperados. Para crearlos necesitamos de la creación de dos ficheros donde se describan las características como los ficheros de entrada, el número de repeticiones que deseamos realizar, los ficheros de salida y el quórum mínimo que necesitamos para asignar créditos. Estos ficheros se ubicarán en el directorio *templates* y los ficheros de entrada y salida se incluirán en el directorio *downloads*.

Para que el servidor comience a ejecutarse necesitaremos ejecutar el binario *bin/start* que controla el arranque de los servicios, así como *bin/stop* que los para.

En este punto sólo quedará que el cliente se registre y pueda ejecutar la *workunit*.

El cliente es más sencillo de instalar, pues instalándolo según la distribución que se tenga ya se puede empezar a utilizar sin problema alguno ni configuración extra.

Las pruebas realizadas son satisfactorias aunque son pruebas sencillas realizadas con proyectos de prueba que BOINC incorpora para comprobar su correcta funcionalidad.

4.2 - Programación de condor_wrapper

BOINC proporciona un sistema para realizar la ejecución de tareas que no incluyen las llamadas a su API. Este programa es llamado *wrapper*.
A través de este programa se realiza la descarga de la aplicación y los ficheros de configuración necesarios. *Wrapper* incluye un sistema que permite especificar los ficheros necesarios que necesita para realizar la ejecución.
En esta especificación se incluye obligatoriamente cual es la aplicación que se desea ejecutar. Discrecionalmente se pueden incluir ficheros que realicen la entrada por teclado para las aplicaciones interactivas, el fichero donde desearemos la salida por pantalla o por error y la línea de comandos necesaria para arrancar la aplicación.

```
<job_desc>
    <task>
        <application>application_to_execute</application>
        [ <stdin_filename>stdin_file</stdin_filename> ]
        [ <stdout_filename>stdout_file</stdout_filename> ]
        [ <stderr_filename>stderr_file</stderr_filename> ]
        [ <command_line>--foo bar</command_line> ]
    </task>
    [ ... ]
</job_desc>
```

Este sistema admite incluir más de una tarea a ejecutar, pero estas tareas son secuenciales. Cuando la secuencia de tareas hubiera acabado, el *wrapper* finalizaría su ejecución.

En el proyecto la aplicación a ejecutar es una aplicación intermedia que nos realiza la configuración y nos ejecuta los procesos necesarios para que Condor encuentre el entorno adecuado y pueda arrancar sin problemas.

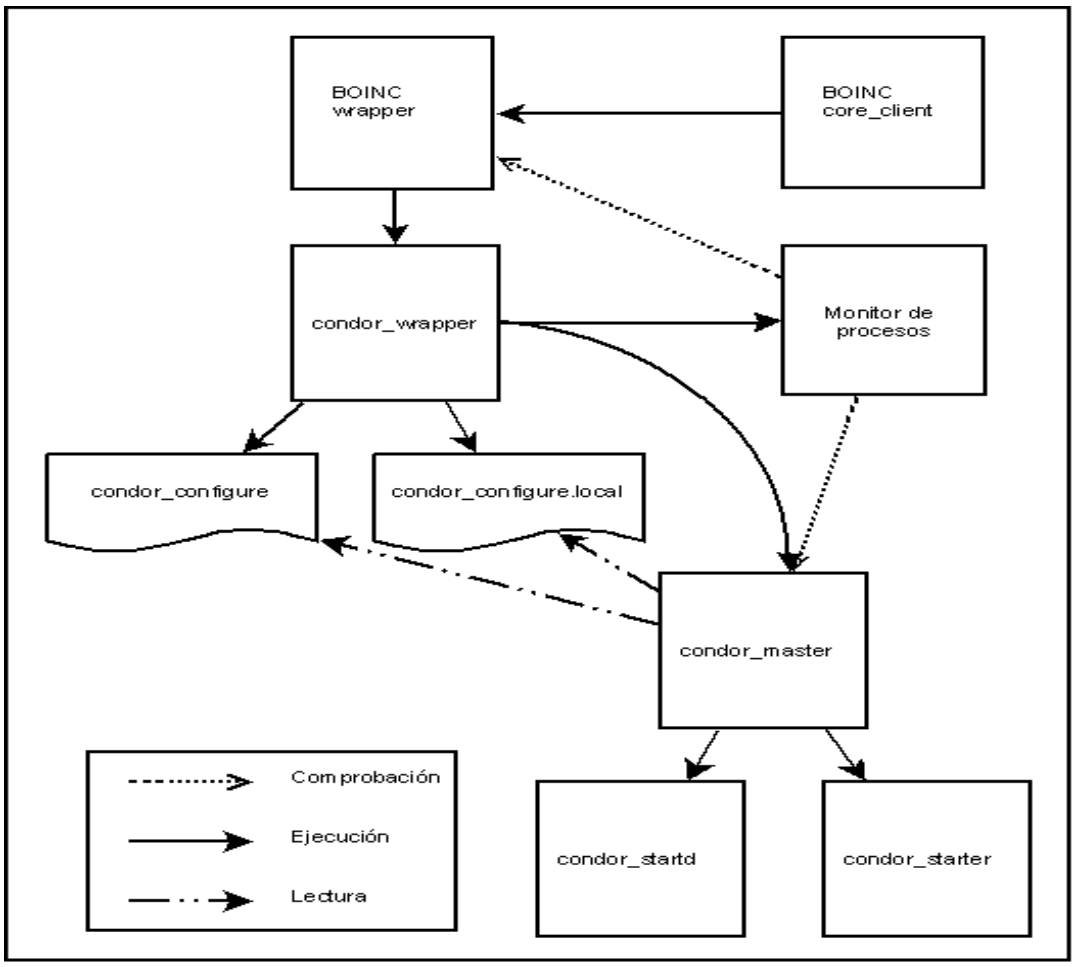

Figura 13 - Estructura de procesos

El proceso *condor_wrapper* es el encargado de arrancar el *daemon condor_master*. Cuando este *daemon* ha arrancado se pierde el control del mismo. Para poder controlarlo y saber el momento en que se debe parar por si el usuario comenzara a hacer uso del ordenador, se ha creado un proceso que monitoriza los procesos encargados de la aplicación.

El monitor de procesos comprueba que el proceso de BOINC *wrapper* no finalice la ejecución. Si esto ocurriera el proceso *condor_wrapper* finalizaría su ejecución y el monitor de procesos lanzaría una señal de finalización al *daemon condor_master* que a su vez pararía los procesos que controla.

Se ha comprobado que en caso de que la aplicación se detuviera, los ficheros no se eliminarían del ordenador a no ser que hubiera una nueva versión, por lo que en la

siguiente ejecución a realizar no se descargarían, ahorrando ancho de banda y agilizando la ejecución.

Una vez arrancada la aplicación se realizan pruebas para comprobar que el monitor de procesos funciona correctamente cuando el ordenador comienza a tener actividad por parte del usuario. Los procesos Condor tardan un poco más en pararse, pero finalmente se paran.

Las mismas ejecuciones que se realizaron para comprobar el buen funcionamiento del sistema Condor se prueban con el sistema completo y el ordenador cliente cambia su estado en Condor al estado ocupado ejecutando un proceso. El resultado se devuelve indicando el ordenador donde se ha ejecutado el proceso, siendo exitoso.

Capítulo 5 - Conclusiones y líneas futuras

Este proyecto explora la posibilidad de la unión de dos sistemas de computación distribuida. Estos sistemas permiten la distribución de tareas entre los diferentes recursos que están conectados al sistema para ejecutarlas, pero cada uno de un modo completamente distinto. Estos dos sistemas de computación distribuida diferentes son Condor y BOINC, pero se intenta que trabajen conjuntamente para alcanzar objetivos comunes.

Condor es un software para la ejecución de trabajos, que crea un entorno capaz de gestionar una gran cantidad de recursos ejecutados durante un largo lapso de tiempo. Puede gestionar *clusters* de ordenadores dedicados, comunicados por una red de área local, utilizando el máximo rendimiento de la capacidad de cómputo disponible que ofrecen los ordenadores.
Condor proporciona una gestión de recursos potente y hace coincidir las propiedades de los productores de recursos con los requisitos de los consumidores de los mismos , adaptándolos al máximo.

Por otro lado, BOINC es una infraestructura de computación distribuida y voluntaria que permite que un usuario pueda instalarse un cliente fácilmente y se registre en proyectos que utilizan la infraestructura BOINC para colaborar con ellos utilizando el tiempo que el ordenador está ocioso.
La instalación del cliente BOINC es sencilla de realizar sea cual sea la arquitectura y el sistema operativo usado y está pensada para que el público en general ayude en los proyectos, facilitándoles el acceso.
Los proyectos suelen ser atractivos para el público y requieren de sistemas de computación masiva que se realiza con la colaboración de voluntarios para poder ejecutar sus tareas.

Cada uno de estos dos sistemas de computación ha sido diseñado con diferentes puntos de vista pero con este proyecto se pretende explorar las posibilidades que existen de aproximar ambos sistemas y conseguir que los sistemas de gestión de *clusters*, como Condor, tengan la posibilidad de incluir sistemas de computación voluntaria como BOINC, para poder ejecutar trabajos controlados por un gestor de recursos seguro y fiable en ordenadores voluntarios de usuarios, los cuales donan sus ciclos de CPU desinteresadamente mediante una infraestructura flexible y llamativa.

El estudio preliminar del proyecto nos ofreció una imagen inicial que nos ayudó a iniciar el proyecto. La aparición de nuevos problemas durante la ejecución del proyecto nos proporcionó un gran estímulo para solucionarlos y llegar a la resolución de cada problema. De esta manera se ha llegado al final del proyecto con los objetivos cumplidos y los módulos propuestos implementados.

El proyecto, además, nos ha dado una visión más general de los sistemas distribuidos pudiendo explorar diferentes sistemas y estudiar soluciones que proporcionasen una respuesta a los problemas.
Otros problemas no se han podido cubrir y quedan a disposición de nuevos desarrolladores.

Las líneas futuras abiertas con este proyecto son las siguientes:

- Condor puede realizar el envío de trabajos a diferentes entornos que llama *universes*. Estos universos gestionan tareas especializadas, como el universo *java*, que permite ejecutar aplicaciones escritas en el lenguaje de programación Java o el universo *paralel* que permite ejecutar tareas paralelas. En el caso de las tareas realizadas con computación voluntaria no disponen de universo, ya que se ha realizado el estudio para tareas que se ejecutan solamente en los universos standard y vanilla. Dos perspectivas se pueden ofrecer: se podría crear un universo que gestionara las tareas que se ejecutaran en entornos voluntarios o se podría modificar el sistema de *ClassAds* de Condor para que las aplicaciones pudieran elegir si desean ser ejecutadas en ordenadores voluntarios. De esta manera se conseguiría diferenciar el tipo de tareas a ejecutar.
- Se podría realizar un estudio del establecimiento de comunicaciones del sistema cuando se encuentran cortafuegos. Condor tiene unos requisitos más estrictos respecto al problema de las comunicaciones. En BOINC es el cliente el que realiza las conexiones con el servidor y utiliza la librería libCurl que tiene soporte, por ejemplo, para los protocolos HTTP, HTTPS y puede tratar con *proxies*. Sería interesante el uso del elemento GCB (Generic Connection Brokering), herramienta del proyecto Condor que ayudaría a resolver el problema.
- El uso de un elemento intermedio entre los dos sistemas, por ejemplo el elemento GCB, modificado pertinentemente, nos permitiría la posibilidad de utilizar aplicaciones paralelas que utilizaran las librerías PVM o MPICH por ejemplo. Este elemento realizaría las funciones de nodo central almacenando las URLs y permitiría que un ordenador pudiera conocer la URL de otro con el que quisiera comunicarse.

Se dejan estas líneas abiertas para la posible realización de nuevos proyectos.

Capítulo 6 - Bibliografía

Distributed Systems. Principles and Paradigms

Andrew S. Tanenbaum y Maarten Van Oteen

Ed. Pearson Prentice Hall. 2a edición 2007

The Grid 2. Blueprint for a New Computing Infrastructure

Ian Foster and Carl Kesselman

Ed. Elservier. 2a edición 2004

High-Performance Computer Architecuture

Harold S. Stone

Ed. Addison-Wesley Publishing Company. 3a edición 1993

Construyendo Aplicaciones Distribuidas con BOINC,

Ciclos perdidos

Revista Linux Magazine nº 23, pág. 48

Condor Manual Version 6.8.5 – Condor Team, University of Wisconsin-Madison

<http://www.cs.wisc.edu/condor/manual/v6.8.5/>

Condor-Users Mail List

<condor-users@cs.wisc.edu>

Página web oficial de BOINC

<http://boinc.berkeley.edu>

Wiki oficial de documentación de BOINC.

<http://boinc.berkeley.edu/trac/wiki>

Unofficial BOINC Wiki
<http://www.boinc-wiki.info/index.php >

BOINC Message Boards
< http://boinc.berkeley.edu/dev/>

Wikipedia, la enciclopedia libre
<http://es.wikipedia.org>

Carlos Moreno Losada

Bellaterra, Junio 2008

RESUMEN

En este proyecto se han visto dos sistemas de computación distribuida diferentes entre ellos: Condor y BOINC. Se exploran las posibilidades para poder conseguir que ambos sistemas logren trabajar conjuntamente, escogiendo la parte más efectiva de cada uno de los sistemas con el fin de complementarse.

RESUM

En aquest projecte s'han vist dos sistemes de computació distribuïda diferents entre ells: Condor i BOINC. S'exploren les possibilitats per aconseguir que ambdós sistemes puguin treballar de forma conjunta, escollint la part més efectiva de cadascun d'aquests sistemes amb la finalitat que es complementin.

ABSTRACT

In this proyect we have seen two different between them distributed Computing systems: Condor and BOINC. We explore our possibilities to let both systems work together, choosing the most effective part of these systems with the aim (obtective-purpose) to complement each other.