UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

INFRAESTRUCTURA PARA COMPUTACIÓN DE ALTA DISPONIBILIDAD Y ADMINISTRACIÓN DE RECURSOS MEDIANTE CONDOR

Tesis presentada para obtener el grado de

Magíster en Redes de Datos

Autor: Ing. Paula Cecilia Martínez

Director: Dr. Carlos García Garino

Co-Director: Lic. Javier Díaz

Julio 2014

Agradecimientos

Agradezco en primer lugar a mis padres y hermano, a mi esposo y mis hijos por la paciencia y apoyo durante estos años.

Un especial agradecimiento a los directivos de la sede Redes y Telecomunicaciones del Instituto Tecnológico Universitario, que me brindó la infraestructura y equipamiento necesarios para realizar los experimentos de esta tesis.

Índice general

Índice de figuras y tablas

Capítulo 1

1. Introducción

En este capítulo se introduce el problema a investigar. Posteriormente se describe el Estado del Arte, dentro del cual se comparan las características de distintos gestores de trabajos, justificando la elección del middleware Condor.

Finalmente, se identifican los objetivos generales y secundarios de la tesis y se presenta la estructura de la misma.

Cabe destacar que la Universidad de Wisconsin-Madison, a partir de octubre del 2012 comenzó a referirse al middleware Condor como HTCondor. El cambio en el nombre del middleware no afecta el uso de las herramientas de la línea de comandos, APIs, variables de entorno o código fuente. Cuando comenzó a desarrollarse esta tesis el middleware se llamaba Condor, por lo que se utiliza este nombre a lo largo de la misma.

1.1 Descripción del problema

En la actualidad la potencia de cálculo es un factor a tener en cuenta en la realización de cualquier actividad que requiera recursos computacionales.

Muchos proyectos necesitan un entorno computacional que brinde una gran capacidad de cómputo durante largos períodos de tiempo.

Gracias al incremento de la potencia de los computadores y a las tecnologías de clustering es posible atacar problemas difíciles de resolver un tiempo atrás (debido al tiempo necesario o al costo de la infraestructura de computación a emplear).

Para acompañar el avance de estas tecnologías de clustering es necesario contar con un middleware que facilite las administración de los recursos computacionales.

Se pueden distinguir los siguientes paradigmas de empleo de un cluster:

- Grid Computing: Este paradigma propone el acceso de recursos de diferente tipo: cómputo, almacenamiento, instrumentos, etc. El mismo se basa en la conformación de Organizaciones Virtuales que comparten recursos. El nombre proviene de una analogía con la Red de Energía Eléctrica (Power Grid) ya que esta tecnología plantea el acceso a recursos computacionales en forma similar a la energía obtenida enchufando un equipo a la red.
- HPC (High Performance Computing): En este paradigma se prima la ejecución lo más rápido posible de las tareas. Conceptos tales como paralelización y

multiproceso entran dentro de este ámbito, y su aplicación directa es hacer que cálculos que pueden durar semanas en un solo equipo se repartan entre varios, dividiendo el trabajo a realizar.

- HTC (High Throughput Computing): En este paradigma se prima la ejecución de la mayor cantidad posible de tareas. Conceptos como gestión de colas y de recursos o workflows son parte de este ámbito. Su aplicación directa pasa por la realización de la mayor cantidad de trabajos a lo largo del tiempo.
- Cloud Computing: Es una forma de compartir recursos disponibles bajo demanda. Los requerimientos de software son brindados como servicios, los cuales están localizados en centros de datos (cloud o clouds) a los cuales se puede acceder sin la necesidad de contar con una infraestructura local (capacidad de cómputo, almacenamiento, etc.).

La tecnología Grid Computing incluye como casos particulares a la Computación de Alto Rendimiento (HPC) y a la de Alta Disponibilidad (HTC). Mientras el paradigma HPC es bastante más conocido en nuestro país existen menos antecedentes para el caso de HTC.

El paradigma HTC puede utilizarse con facilidad cuando se tiene una multitud de cálculos de tamaño mediano / pequeño, y se quiere agilizar la gestión de todos ellos.

Para llevar a cabo el procesamiento de tareas que requieren gran poder de cálculo se suele recurrir a lo que hoy día llamamos “supercomputadoras” que, con una capacidad de procesamiento muy superior a la de las computadoras de escritorio, son capaces de ejecutar cientos de tareas en un tiempo relativamente corto.

El principal problema de estos súper-computadores es el costo económico, ya que este hardware presenta un precio muy elevado que no todas las instituciones pueden abordar, el cual además es dedicado. Para solventar el obstáculo económico, en los últimos años han comenzado a utilizarse soluciones que requieren menos inversión: los llamados entornos de alto rendimiento computacional HTC (High-Throughput Computing).

El acceso a equipamiento informático con gran capacidad de cómputo ha sido un objetivo perseguido por los científicos durante décadas. Es en los años setenta cuando se comienza a utilizar ese poder de cómputo a bajo costo, mediante una colección de pequeños dispositivos, en lugar de una única y costosa súper-computadora. Sin embargo, la computación distribuida presenta algunos inconvenientes como la pérdida de los mensajes o que éstos lleguen demorados o corruptos.

Para construir un sistema relativamente controlable se necesita de algoritmos precisos, lo que no sucedió a comienzos de los ochenta cuando algunos sistemas evidenciaron la tensión existente entre la consistencia y la disponibilidad. Es en este contexto que surge el proyecto Condor [1-3] en la Universidad de Wisconsin, como un sistema para computación distribuida en contraste con el modelo centralizado dominante.

Condor es un middleware diseñado para Computación de Alta Disponibilidad (High Throughput Computing HTC) y empleo de recursos ociosos. Este permite de manera transparente y simultánea explotar las capacidades de estaciones de trabajo que pueden estar distribuidas en el mundo y pertenecer a distintos individuos, grupos,

departamentos e instituciones. La idea principal del middleware es hacer posible que cálculos con alto costo de gestión puedan ser realizados de forma ágil y eficiente.

Condor se presentó como un sistema único en el sentido de que cada participante tenía la libertad de contribuir, ya sea mucho o poco, con capacidad de procesamiento.

En los años noventa se produce un enorme crecimiento en el campo de la computación distribuida y surgen una importante variedad de sistemas de ejecución por lotes tales como LoadLeveler, LSF, Maui, NQE y PBS.

Algunos sistemas de cómputo distribuido como SETI@Home y Napster concientizaron al público en general sobre el poder de la computación distribuida. Comienza entonces a surgir la idea de Grid Computing como una forma de compartir recursos traspasando los límites de la organización. Durante este período Condor incorpora funcionalidades para dar soporte y contribuir con nuevos estándares como PVM, MPI y Java.

A medida que los científicos agrupaban capacidad de cómputo conformando grids utilizando protocolos como Grid Resource Access and Management (GRAM), Grid Security Infrastructure (GSI), y GridFTP, incluidas en Globus. Condor y Gloubus incorporan mecanismos de comunicación para resolver aspectos tales como seguridad, administración de recursos y ejecución de trabajos.

Condor, incorpora tecnologías computacionales y protocolos emergentes basados en Grid y Cloud computing.

Actualmente, la CERN está utilizando el middleware en un cluster de 4000 núcleos para análisis de colisiones. Otro proyecto, en el cual se ejecutan modelos estocásticos sobre el middleware, combina las funcionalidades que ofrece Condor con Cloud computing. [4].

1.2 Importancia y Motivación

Los trabajos que requieren capacidad de cómputo intensiva necesitan de un administrador de carga de trabajo especializado, que brinde mecanismos de cola, políticas de planificación, esquema de prioridades, monitoreo y administración de recursos.

Cuando los usuarios emiten sus trabajos, el administrador deberá decidir cuándo y dónde ejecutarlos, teniendo en cuenta sus requerimientos, realizar un monitoreo del progreso e informar al usuario cuando haya finalizado su ejecución.

Al integrar capacidad de proceso, almacenamiento y acceso a recursos remotos, se podrán ejecutar aplicaciones que no pueden procesarse en una computadora única y así satisfacer demandas de cómputo complejas.

En este trabajo se discute el uso de Condor como gestor de recursos disponibles en un entorno de Computación de Alta Disponibilidad ya que es un sistema que ofrece funcionalidades de HTC donde los usuarios no tienen que preocuparse, por ejemplo, dónde enviar sus trabajos para ejecución, ni de tener que enviar manualmente un gran número de ellos cuando así lo requieran.

Mediante un fichero de configuración (muy sencillo), Condor se encarga de realizar todas estas tareas, facilitando enormemente la gestión de dichos cálculos.

Condor aunque está diseñado principalmente para correr en un conjunto de nodos dedicados (un cluster, por ejemplo), es capaz de correr en estaciones no dedicadas sin ningún tipo de problema, aprovechando los "ciclos muertos" de dichos equipos, haciendo que el conjunto sea altamente potente y escalable.

1.3 Estado del Arte

En esta sección se analizan las principales características de los gestores de trabajos que constituyen el Estado del Arte.

La utilización de Condor ha dado muy buenos resultados en entornos HTC con recursos computacionales distribuidos.

Entre los gestores de trabajos disponibles se pueden mencionar:

- Legion
- LSF
- PBS / Torque
- Oracle Grid Engine
- Unicore
- Slurm
- Condor

A continuación se desarrollarán los aspectos más relevantes de estas herramientas.

1.3.1 Legion

Legion [5] es un middleware basado en objetos diseñado para acceder a una gran cantidad de recursos de manera segura y transparente. Este proyecto fue creado por National Science Foundation entre otras agencias y desarrollado en la Universidad de Virginia, en el año 1993. Posteriormente los desarrolladores de Legion conformaron la corporación Avaki la cual abandonó el proyecto en el año 2005. En el año 2005 Sybase adquirió Avaki pero a partir del año 2010 no se brinda mas soporte.

La idea detrás de Legion es que los usuarios accedan a una gran cantidad y variedad de recursos pero teniendo la visión de una única computadora, pudiendo crear espacios

virtuales compartidos para colaborar en proyectos de investigación e intercambiar información.

Este toolkit basado en objetos proporciona las infraestructuras necesarias para que un conjunto heterogéneo de máquinas distribuidas geográficamente funcionen de forma coordinada, presentando al usuario una sola máquina virtual.

Legion se ubica por encima del sistema operativo del usuario, actuando como un coordinador entre el host y los recursos solicitados. Los usuarios pueden proteger sus recursos de otros usuarios y los administradores pueden implementar políticas para definir los recursos a los que podrá acceder un determinado usuario.

Legion implementa un sistema de nombres controlado por el usuario llamado context space para que puedan crear y utilizar objetos en sistemas diversos, y ejecutar aplicaciones escritas en múltiples lenguajes.

Legion brinda mecanismos para autenticación, autorización y para conservar la integridad y confidencialidad de los datos. Los usuarios se autentican con un nombre de usuario y contraseña y el sistema crea un objeto autenticación para cada usuario. Este objeto contiene la clave privada del usuario, la contraseña encriptada e información del perfil del usuario y es almacenado en el disco local. Cuando el usuario escribe su contraseña al momento de logearse en el sistema, se compara la contraseña escrita con la almacenada en el objeto autenticación. Luego de que el usuario se ha autenticado, el objeto autenticación genera y firma una credencial X.509. Las credenciales se almacenan y son protegidas por el sistema operativo y son utilizadas para acceder a los recursos de la Grid por ejemplo mediante Globus.

En términos de autorización, Legion es bastante flexible. Por defecto se crean Listas de Control de Acceso (ACLs), y mediante la especificación de Allow/Deny se indica quienes pueden realizar una determinada acción sobre el sistema y quiénes no.

Re0specto a la integridad y confidencialidad de los datos, Legion permite tres modos de operación para acceder a los datos: privado, protegido y ninguno. En el modo privado, los datos son completamente encriptados, en el modo protegido, el emisor de los datos realiza un control, por ejemplo MD5 del mensaje actual. En la tercera modalidad, los datos se transmiten en texto plano, excepto por las credenciales. Legion utiliza OpenSSL para encriptación.

Legion confía en los mecanismos de seguridad del sistema operativo en lo que se refiere al almacenamiento de los datos.

Legion no ofrece una solución para hacer uso eficiente de pools de discos y sistemas de archivos de alto rendimiento.

Respecto a la administración de recursos, Legion presenta tres tipos de recursos: hosts (recursos computacionales) y values (recursos de almacenamiento) y recursos de red. El modelo de planificación de Legion consiste en tres componentes principales: una base de datos con información del estado de los recursos, un módulo el cual administra solicitudes (objetos) para asociarse con los recursos (hosts y values), y un agente de activación responsable de implementar la planificación. Estos componentes se llaman respectivamente Collection, Scheduler y Enactor, respectivamente.

La capa Collector interactúa con los objetos tipo recursos para recolectar información de estado que describa el sistema, luego la capa Scheduler solicita a la capa Collection que

defina un conjunto de recursos disponibles que puedan asociarse con los requerimientos del Scheduler. Luego de que se ha realizado la planificación, el Scheduler envía una lista a la capa Enactor para la implementación de la planificación.

Esta capa realiza reservas con los recursos y reporta el resultado al Scheduler. Una vez que el Scheduler lo aprueba, el Enactor ubica los objetos en los hosts y monitorea su estado.

Este enfoque totalmente orientado a objetos dificulta la interacción con algunas aplicaciones y en particular con aplicaciones paralelas escritas en lenguajes nuevos o adaptados, como Mentat o Basic FORTRAN Support (un conjunto de directivas para FORTRAN).

Legion es un sistema completamente integrado e interactúa con mecanismos industriales para la distribución de sistemas orientados a objetos. Legion sustituye algunos servicios del sistema operativo, por lo que habrá algunas restricciones para ciertas aplicaciones.

Como se menciona al comienzo de este apartado, Legion no tiene soporte desde el año 2010.

1.3.2 LSF – Load Sharing Facility

Se trata de una suite comercial de productos para administración de recursos que planifica, monitoriza y analiza la carga de trabajos de una red de las computadoras. Dentro de la suite se incorpora un sistema de colas llamado LSF batch. LSF [6] es un sistema de pago (propietario) desarrollado por la compañía Platform.

El administrador de LSF posee herramientas gráficas, además de la línea de comandos, desde las cuales puede llevar a cabo la configuración del sistema de forma sencilla y completa. El administrador tendrá un control total sobre usuarios, recursos y trabajos.

El usuario, al igual que el administrador, dispone de una serie de herramientas gráficas (además de la clásica línea de comandos) que le serán de gran utilidad a la hora de remitir sus trabajos (que a diferencia del PBS, no se realiza con macros), es en este momento cuando se especifican los recursos que va a utilizar un trabajo. Una vez comenzada la ejecución, el usuario puede controlar su trabajo de la misma manera. Por otro lado la información final es devuelta al usuario o bien en forma de e-mail o bien redirigida a un fichero, ofreciendo tanto la salida que produce el trabajo como información sobre los recursos usados por éste.

En lo que se refiere al soporte de aplicaciones paralelas, LSF se comporta como un “asignador” de recursos, por ejemplo, en una aplicación MPI, la ejecución se va a limitar al número de procesadores solicitados previamente, por lo que los procesos hijo estarán limitados a esos procesadores y LSF será capaz de controlar los recursos consumidos por estos procesos.

Por otra parte, LSF soporta checkpointing tipo kernel (en los sistemas operativos que lo proporcionen) y de nivel de usuario, siendo necesario para este último la recompilación de las aplicaciones con librerías especiales.

El sistema además soporta migración de procesos y también balanceo de carga. La distribución ofrece varios schedulers, pudiendo coexistir múltiples políticas de scheduling en un mismo sistema. En cuanto a tolerancia a fallos, las caídas de la red son soportadas, así como las caídas del nodo master.

LSF es un sistema de pago, con y como es habitual en estos sistemas el código fuente no está disponible. LSF ha sido diseñado para establecerse por encima del sistema operativo existente, lo que le hace muy portable (soportado por casi todos los sistemas UNIX y por Windows NT/2000).

1.3.3 PBS – Portable Batch System

Se trata de un sistema de colas desarrollado en sus comienzos por Veridian Systems para la NASA a mediados de los noventa, porque requería un software para el manejo de recursos, siendo su actual empresa propietaria Altair. Posee dos versiones: una paga, llamada PBSPro [7] y una segunda de libre distribución, denominada OpenPBS Version 2.3.2 [8]. Esta versión gratuita es para uso académico y Veridian se ha focalizado en el desarrollo de PBSPro, por lo que el futuro de OpenPBS es incierto.

Es una herramienta de software que se creó con el objetivo fundamental de solucionar el manejo de carga de trabajo para sistemas de computación de alto rendimiento y para manejar clusters de computadoras Linux y Unix.

PBS permite definir e implementar una política para la ejecución de los trabajos, como definir qué tipos de recursos y cuanto de cada recurso puede ser usado por los trabajos. Además, proporciona un mecanismo que el usuario puede usar para asegurar que un trabajo tenga acceso a los recursos requeridos para ejecutarse. PBS está formado por varios componentes como el servidor y el cliente, que funcionan estableciendo una interacción entre ellos. El cliente realiza una solicitud a un servidor y el servidor se encarga de realizar el trabajo del cliente. El servidor de PBS proporciona servicios de crear, ejecutar, modificar o eliminar trabajos del cliente, dependiendo de su solicitud.

Para llevar a cabo el control del sistema (usuarios, recursos y trabajos) el administrador posee una herramienta gráfica de acceso exclusivo.

El manejo del sistema por parte del usuario se puede hacer de dos formas distintas: desde la línea de comandos o a través del entorno gráfico que se proporciona con el paquete. La sumisión de trabajos se hace a través de macros, shell scripts, en las que se especificarán tanto las características del trabajo, como los recursos que necesitará para su ejecución. Una vez comenzada la ejecución, el usuario podrá controlar (borrar, priorizar, cambiar recursos asignados, etc.) sus trabajos con libertad.

La información relativa a la ejecución (tiempos, utilización de recursos, etc.) se obtiene vía e-mail. Los resultados del trabajo pueden ser redireccionados a un fichero de salida.

Respecto a los requisitos previos, se trata de un sistema de distribución gratuito donde se ofrece el código fuente. Opera en entornos de red UNIX multiplataforma, incluyendo clusters homogéneos y heterogéneos de estaciones de trabajo, supercomputadores y sistemas masivamente paralelos.

No necesita ni hardware ni software adicional, no se requieren sistemas de ficheros en red como AFS o NFS, ya que las transferencias se realizan vía RCP o SCP.

La primera peculiaridad de este sistema es que adjudica un nodo en exclusiva a cada trabajo (spacesharing), independientemente del número de procesadores que existan o de la cantidad de memoria disponible. Si el trabajo es paralelo, el usuario debe especificar cuantos y qué tipo de nodos se requieren, esto se hace a través de una herramienta mediante la cual se puede crear, monitorizar y controlar tareas en nodos remotos. Sin embargo, esta herramienta es muy poco utilizada, ya que los programadores en PVM y MPI prefieren usar las utilidades de las propias librerías.

Una segunda característica es la posibilidad de hacer checkpoint, pero en este caso de forma muy limitada, ya que sólo se contempla en sistemas IRIX 6.5 o superiores, en este caso el checkpoint es a nivel kernel, por lo que no es exactamente el sistema de colas quien lo lleva a cabo. Por otra parte, en PBS no se contempla la migración de procesos, pero si el balanceo de carga que se hace de forma automática si se especifica en la configuración.

En cuanto a la tolerancia a fallos, el sistema es bastante limitado; los trabajos que se están ejecutando en un nodo que se cae, se pierden. Por otro lado, una caída en el nodo servidor (donde residen los demonios del sistema) hará imposible la recuperación de los trabajos que estaban en ejecución. La distribución viene con diferentes políticas de scheduling, y se ofrece la posibilidad de crear nuevas políticas. La que se instala por defecto es FIFO.

La principal desventaja de OpenPBS es el futuro incierto del desarrollo de la versión open source. Otra desventaja es que la interfaz gráfica de PBS sólo permite trabajar visualizar los trabajos en ejecución y monitorear el uso del sistema por parte de los clientes. La configuración del sistema debe realizarse por línea de comandos y mediante archivos de texto.

Otra desventaja es que PBS no representa los trabajos como colas, por lo que no es posible definir políticas complejas.

OpenPBS no tiene soporte para funcionalidades adicionales como sí lo tiene PBSPro, como ser un módulo planificador avanzado, integración con servicios Globus y soporte para Windows.

El administrador de recursos Torque [9] está basado en el proyecto OpenPBS versión 2.3 pero incorpora mejoras tales como tolerancia a fallos, mejoras en la escalabilidad, en el sistema de planificación de trabajos, entre otras. A fin de mejorar las prestaciones de Torque, suele integrárselo con el planificador de cluster Maui o con el administrador de trabajos Moab, el cual es pago.

1.3.4 Oracle Grid Engine

Oracle Grid Engine [10] es actualmente una de las herramientas comerciales para administración de carga de trabajo más utilizadas. Ofrece la posibilidad de adaptarse a distintos entornos computacionales y distintas cargas de trabajo.

Esta herramienta permite implementar políticas de scheduling avanzadas y reserva de recursos y adapta automáticamente el tamaño del cluster basándose en objetivos de nivel de servicios.

OGE permite compartir recursos entre dos clusters OGE con el fin de aumentar la utilización del centro de datos, pudiendo inclusive hacer uso de servicios de una nube pública o privada cuando sea necesario.

Oracle llama a esta facilidad cloud bursting, la cual permite a la organización administrar tráfico regular y picos de carga de trabajo impredecibles sin necesidad de invertir en recursos adicionales.

La última versión de OGE es Oracle Grid Engine 6.2 actualización 6 incluye funcionalidades avanzadas, como por ejemplo hacer uso de recursos provistos por Amazon EC2.

Otra característica de esta versión es la mejora en el mecanismo de preferencias de ejecución de los trabajos, resultando en un aumento en la utilización del centro de datos de la organización.

Esta nueva actualización incluye la posibilidad de especificar límites en la ejecución de trabajos que emiten los usuarios. Para el caso de usuarios que emiten grandes trabajos paramétricos, el planificador de OGE les permite compartir recursos de manera más equitativa, lo que resulta en una mayor eficiencia y menores tiempos de ejecución.

Respecto a las plataformas de ejecución, el nodo maestro de OGE soporta varias versiones de Solaris, Linux y Windows, entre las más destacadas.

1.3.5 Unicore

UNICORE (UNiform Interface to COmputer REsources) [11] es un software open source que proporciona a los usuarios una interfaz gráfica para la preparación de trabajos y acceso seguro e intuitivo a los recursos de computación. Oculta las particularidades de cada sistema para facilitar el desarrollo de aplicaciones distribuidas.

UNICORE está diseñado para trabajo por lotes. Un trabajo en UNICORE consiste en un conjunto de tareas, con dependencias que indican relaciones temporales o transferencias de datos. La petición de ejecución de un trabajo indica dónde se deben ejecutar las tareas que la forman, y qué recursos requieren. Las tareas también se pueden dividir en subtareas, creando de este modo una jerarquía por la que diferentes partes de un trabajo se pueden ejecutar en distintos sistemas.

Los objetivos de diseño de UNICORE incluyen: un interfaz de usuario uniforme y fácil de usar, una arquitectura abierta basada en el concepto de tareas abstractas, una arquitectura de seguridad consistente, y mínima interferencia con los procedimientos administrativos locales.

UNICORE es un sistema cliente-servidor con una arquitectura dividida en tres niveles, que son:

- Nivel de usuario: contiene un cliente Java, compuesto por dos componentes:

    - JPA (Job Preparation Agent): se utiliza para el desarrollo de las tareas ejecutadas.
    - JMC (Job Monitor Component): mediante el cual se obtienen el estado y los resultados de las tareas.
    - Las tareas, los comandos y los resultados están formulados utilizando AJO (Abstract Job Object)

- Nivel de Server: contiene los gateways, que comunican los clientes con los recursos Grid (organizados en Unicore sites o Usites)

- Nivel de Target system: compuesto por los Usites, que ofrecen recursos computacionales o de almacenamiento agrupados y organizados dentro del Usite en virtual sites (Vsites). Los Vsites están formados por:
    - Múltiples instancias de supervisores de tareas o NJS (Network Job Supervisors), que controlan la ejecución de las tareas y llevan a cabo las funciones de autenticación mediante las UUDB (UNICORE User Data Base) Múltiples instancias de Interfaces de Target System, que controlan diferentes recursos y subsistemas.

Las funciones básicas de UNICORE son:

- Desarrollo de tareas de usuario para ser ejecutadas en sistemas UNICORE
- Gestión de tareas: provee de un control total sobre las tareas al usuario
- Tratamiento de datos: control sobre los datos que deben ser importados, exportados o transferidos a otros Usites.
- Soporte de aplicaciones: el interfaz de usuario contiene plug-ins para datos de aplicaciones científicas
- Control de flujo: descripción de las tareas con grafos
- Firma única
- Descubrimiento de recursos

A nivel de seguridad, UNICORE utiliza una base de datos donde se asocian las identidades de los usuarios a las cuentas locales.

Un aspecto importante de los middleware para Grid, es el descubrimiento dinámico de recursos, característica que el middleware UNICORE no soporta.

1.3.6 Slurm - Simple Linux Utility for Resource Management

Slurm [12] es un es un planificador de tareas open-source usado actualmente por muchos de los super ordenadores a nivel mundial y por clusters de computación. Ofrece tres funciones principales. En primer lugar permite el acceso exclusivo / no exclusivo a los usuarios a recursos computacionales durante una cierta cantidad de tiempo a fin de que ejecuten sus trabajos. Segundo, brinda un entorno para inicializar, ejecutar y monitorear trabajos (típicamente trabajos paralelos) en un conjunto de nodos. Finalmente, arbitra posibles contenciones por recursos mediante la administración de colas de trabajos pendientes.

Slurm está disponible para varias plataformas del sistema operativo Linux. De acuerdo al sitio oficial de Slurm, no está disponible para Windows, y para otras plataformas, como FreeBSD, NetBSD, OS X y Solaris, está disponible a partir de determinadas versiones.

1.3.7 Condor

Condor es un entorno distribuido open source (desde el año 2008) diseñado para Computación de Alta Disponibilidad (High Throughput Computing HTC) y empleo de recursos ociosos (CPU Harvesting), que se basa en emplear de manera cooperativa equipos (escritorio, laboratorios, etc.) disponibles. Condor implementa muchas facilidades como soporte para lanzar arreglos de trabajos, o instanciar workflows en casos más complejos.

Además es capaz de manejar tolerancia a fallos, migración de tareas, replanificación de tareas e incluso de resguardar procesos en curso (checkpointing) para su eventual reejecución. Un gestor o broker (Condor Matchmaker) se encarga de planificar las tareas y enviarlas a los recursos disponibles para su ejecución. Condor-G [1-3] es la extensión de Condor al entorno de Grid y simplemente ejecuta trabajos, pero requiere trabajo adicional para la planificación de tareas.

El middleware Condor permite de manera transparente y simultánea explotar las capacidades de estaciones de trabajo que pueden estar distribuidas en el mundo y pertenecer a distintos individuos, grupos, departamentos e instituciones.

Condor permite implementar mecanismos para autenticación y autorización de usuarios y para la encriptación e integridad de los datos, aspecto a ser investigado con detalle para conocer la robustez de los esquemas de seguridad contemplados en Condor. Para fortalecer estos mecanismos Condor interactúa con GSI, protocolo de seguridad implementado en Grid.

Una de las ventajas fundamentales del middleware es que, aunque esté diseñado principalmente para ejecutarse en un conjunto de nodos dedicados (un cluster, por ejemplo), es capaz de ejecutarse en estaciones de trabajo no dedicadas.

La idea principal de Condor es facilitar las tareas de los investigadores creando un entorno de “lanza y olvídate”, haciendo posible que cálculos con alto costo de gestión puedan ser realizados de forma ágil y eficiente.

De la misma forma, Condor puede interactuar de forma sencilla en el Grid computing, aprovechando la capacidad de todos los recursos informáticos de la red y aumentando la capacidad de cálculo del sistema.

Algunas de las características más interesantes de Condor son las siguientes:

- Sistema de colas integrado (el trabajo se lanza en Condor, y él busca el nodo más libre y lo ejecuta).
- Capacidad de ejecutar una simulación numerosas veces sin tener que relanzar a mano el cálculo.
- Notificación de finalización del cálculo vía e-mail.
- Posibilidad de realizar selección (es posible decir “solo lances el cálculo en máquinas con más de 512 MB de RAM).
- Posibilidad de preferencias (“lanza el cálculo en la máquina que tenga más HD”).
- Sistema de prioridades sencillo (en el usuario y en el administrador).
- Configuración muy sencilla (trabaja una vez, y reusa).
- Funciona en nodos no dedicados, sin molestar al usuario.
- Disponible en Windows y Linux (es posible dentro de una configuración que el trabajo se ejecute en multiplataforma).
- Posibilidad de checkpointing (se guarda cada cierto tiempo el estado del cálculo).
- Conexión a Globus directa: Es posible emplear la potencia de cálculo de otros cluster mediante Condor-G.
- Posibilidad de ejecución condicional.

Condor realiza dos funciones principales: Procesado de colas y gestión de recursos. El procesado de colas consiste en recoger las peticiones de trabajos y ponerlas de forma priorizada en una cola para su ejecución, mientras que la gestión de recursos se encarga de saber qué equipos están activos y con qué características cuentan, así como de repartir los trabajos de forma óptima.

Para recoger los trabajos Condor emplea un fichero de configuración que tiene todos los parámetros necesarios del mismo (programa a ejecutar, ficheros de configuración, ficheros de salida, variables de entorno, etc.). Dicho fichero de configuración debe de ser generado por el usuario mediante una sintaxis sencilla y potente (existen ficheros base explicados, así como ejemplos para las configuraciones más frecuentes).

Condor recoge todos los trabajos a ejecutar y examina los recursos existentes en el sistema. Si hay equipos suficientes para todos los trabajos los ordena y envía directamente. Si no hay equipos suficientes, los ordena según prioridades (Condor tiene por defecto un sistema de “reparto justo de carga” que asegura que todos los usuarios tienen una parte igual de los recursos, aunque es posible asignar diversas prioridades), y los va ejecutando a medida que quedan equipos libres.

La gestión de recursos se realiza mediante unas etiquetas asignables tanto a los trabajos como a los recursos, que se denominan ClassAds y especifican las características de unos y otros. Una vez realizada esta caracterización, la gestión de recursos únicamente consiste en emparejar ClassAd de trabajos con ClassAd de recursos.

Una vez Condor decide ejecutar un trabajo, transfiere tanto el programa como los ficheros necesarios para su ejecución al equipo encargado de realizarlo y comienza su ejecución. Los resultados (así como un fichero de log en el que se refleja el progreso de la tarea) se transfieren de nuevo al equipo del usuario.

Condor además, permite el checkpointing, que consiste en guardar de forma periódica el estado completo de una tarea para poder recuperarla a posteriori.

1.4 Análisis de los middleware

En este capítulo se ha planteado el problema de la administración de recursos en entornos de computación de alta disponibilidad y se han presentado distintas herramientas para administrar dichos entornos. Muchas de estas herramientas presentan inconvenientes y limitaciones, que se resumen a continuación: Legion no ofrece soporte desde el año 2010. LSF es un sistema de pago, con y como es habitual en estos sistemas el código fuente no está disponible. PBS ofrece una versión paga y la versión libre ha derivado en Torque, el cual para mejorar sus prestaciones necesita vincularse con el planificador de cluster Maui o con el administrador de trabajos Moab, el cual es pago. Oracle Grid Engine es actualmente una de las herramientas para administración de carga de trabajo más utilizadas, pero es paga. UNICORE no ofrece el descubrimiento dinámico de recursos, aspecto importante de los middleware para Grid.

Como característica común a estos middleware se puede mencionar que todos ofrecen similitudes a la hora de gestionar colas. El archivo para enviar trabajos para ejecución en Condor, Slurm y Torque son muy similares, pero Condor es más flexible y personalizado a la hora de especificar los recursos sobre los cuales quiere ejecutarse el trabajo.

Otra característica común que puede mencionarse es que Globus, además de la conectividad con Condor, también permite la conectividad con LSF, Torque, Oracle Grid Engine y UNICORE.

Como se mencionó anteriormente, Condor posee características similares a otros middleware en cuanto al sistema de manejo de colas, pero se diferencia notablemente al poder ser utilizado para administrar cluster de recursos dedicados, pool de recursos con disponibilidad parcial e infraestructuras mixtas. Otro aspecto relevante de este middleware es la alta flexibilidad del mecanismo de asociación de recursos con trabajos,

además de ofrecer al usuario mecanismos de encolamiento, políticas de planificación, esquemas de prioridades, monitoreo y administración de recursos.

Del estudio de las distintas herramientas surge que el middleware Condor, relativamente poco conocido en nuestro país, es una herramienta útil y de bajo costo que pone a nuestra disposición toda la capacidad de cálculo ociosa en nuestra red, permitiendo incrementar considerablemente los recursos computacionales disponibles.

1.5 Objetivos

El objetivo general de la tesis es investigar las capacidades y empleo de Condor para administrar de manera eficiente un entorno de Computación de Alta Disponibilidad.

Los objetivos parciales que abordará esta tesis son:

- Estudiar las características y requerimientos de los entornos de Computación de Alta Disponibilidad.
- Investigar posibilidades de interacción entre Condor y Grid.
- Estudiar los esquemas de seguridad disponibles en Condor.
- Implementar mecanismos y políticas de seguridad para Condor.
- Analizar los requerimientos y características de los trabajos para que sean ejecutados sobre recursos Grid bajo Condor.
- Realizar pruebas de funcionamiento de Condor en entornos LAN campus y extra campus.

Con este trabajo se pretende realizar un aporte a los entornos computacionales de alta disponibilidad investigando los mecanismos, políticas y estrategias que implementa el software Condor para la administración eficiente de recursos en dichos entornos.

1.6 Estructura del Trabajo

En el capítulo 2 se discute el despliegue, instalación, conectividad y configuración de un pool de Condor. Se describen también sus componentes y funciones. También se destaca la capacidad de Condor de administrar recursos dedicados, no dedicados e infraestructuras mixtas. Se describen las metodologías implementadas en esta tesis para instalar un pool del middleware. Se presenta la infraestructura disponible para la ejecución de los experimentos computacionales desarrollados y se detallan los mecanismos de interacción de Condor con la computación grid.

En el capítulo 3 Se discuten los mecanismos de seguridad que se implementan en un entorno Condor.

En el capítulo 4 se plantean la gestión de recursos, la administración de trabajos en Condor, el envío, ejecución y monitoreo de los trabajos, el proceso de matchmaking, las prioridades, el mecanismo de checkpointing y workflows.

En el capítulo 5 se desarrollan los experimentos computacionales sobre la infraestructura presentada en el capítulo 2 y se discuten sus resultados. Los experimentos se desarrollan para aplicaciones de Sistemas de Detección de Intrusos (IDS), Sodge y MPI.

Finalmente en el capítulo 6 se presentan las conclusiones y los trabajos futuros.

Capítulo 2

2. Descripción de un pool de Condor: despliegue, instalación, conectividad y configuración

En el presente capítulo se describen los componentes de un pool de Condor y sus funciones y se destaca la capacidad de Condor para administrar recursos dedicados, no dedicados e infraestructuras mixtas.

También se reseñan características destacadas de Condor, que pueden consultarse en referencias acerca del middleware [1-4] y trabajos previos de los autores [13,14]. A grandes rasgos corresponde decir que Condor convierte a un conjunto de recursos de cómputo en una computadora distribuida, adecuada para el procesamiento masivo de trabajos lotes. De esta manera se puede agregar valor a recursos de disponibilidad parcial como es el caso de laboratorios de enseñanza, por ejemplo, o incluso máquinas de escritorio.

El capítulo comienza en la sección 2.1 con una descripción de los componentes de un pool de Condor, sus funciones y características. Luego en la sección 2.2, se presentan dos metodologías para instalar un pool de Condor. En la sección 2.3 se describe la infraestructura disponible para la ejecución de los experimentos computacionales, destacando la capacidad de Condor para procesar aplicaciones sobre recursos dedicados, no dedicados y sobre infraestructuras mixtas. En esta sección también se indican los pasos para instalar Condor en un pool de recursos computacionales. En el capítulo 5 se discuten en detalle dichos experimentos.

En la sección 2.4 se detallan los mecanismos de interacción de Condor con la computación grid, implementando entre otros, un experimento computacional donde se muestra la interacción de Condor con Globus, una de las herramientas actuales más poderosas en la conformación de grids computaciones.

2.1. Descripción de un pool de Condor

El propósito principal de un sistema distribuido es que los usuarios ejecuten sus trabajos en un pool de recursos compartidos. Un pool se puede definir como un número arbitrario de nodos, con distintas arquitecturas y con sistemas operativos diversos que se conectan mediante una red. Conceptualmente se dice que un pool de Condor es una colección de recursos y de peticiones de recursos [1].

Un componente importante de un pool de Condor es la figura del Administrador del mismo, el cual se encarga de configurar el pool, administrar los recursos y los usuarios del mismo como se explica en el capítulo 3 de esta tesis.

Las actividades claves de Condor como ser la asignación de trabajos a recursos, inicio y ejecución de trabajos y posterior recolección y devolución de datos se mantienen separadamente a fin de permitir la modularización de Condor en componentes claramente definidos. Esta descomposición de la arquitectura de Condor está basada en demonios que se distribuyen entre el sitio de emisión del trabajo, el administrador central y el sitio de ejecución del trabajo.

En un pool de Condor solo puede haber un administrador central, pero puede haber múltiples sitios para la emisión y ejecución de trabajos y un mismo nodo puede cumplir múltiples funciones. Los roles que puede cumplir un nodo en un pool de Condor son:

- Administrador Central: Existe un administrador central el cual es único en cada pool de Condor. Este equipo es el responsable de recopilar la información del pool y de sus recursos y de asociar los requerimientos de los trabajos con los recursos adecuados. Este es un proceso de matchmaking o asociación, el cual es simétrico; tanto los requerimientos de los trabajos como de los recursos son tenidos en cuenta por el administrador central a la hora de asociarlos.

  El nodo que realiza el papel de administrador central desempeña una función muy importante en un pool de Condor, por tanto, dicho nodo debe ser muy fiable. Se debe elegir como administrador central a un nodo que esté continuamente activo, o al menos que pueda ser reiniciado rápidamente si algo no funciona correctamente.

  Otro requisito que debe tener este nodo es el de poseer una buena conexión en la red hacia todos los nodos del pool. Es decir, toda la información del pool residirá en el administrador central, y este nodo será el centro de peticiones de recursos que haya por parte de cualquier nodo del pool.

  En el administrador central residen dos importantes demonios de Condor, el demonio *Negotiator* y el demonio *Collector*.

  El demonio *Collector* recoge los ClassAds (que especifican las caracterísitcas de los recursos y los trabajos) y determina el estado de todos los nodos del pool de Condor. Con esta información y durante un ciclo regular de negociación, el demonio *Negotiator* se ocupa de asociar los recursos con solicitudes de ejecución de trabajos. Este demonio también se encarga de registrar las estadísticas del pool y de administrar las prioridades del usuario de acuerdo al uso actual del recurso. Por ejemplo, una asociación previa de un trabajo a un recurso puede ser revocada si un trabajo y/o un usuario de mayor prioridad solicitan un recurso particular.

- Nodo de ejecución: además del administrador central del pool de Condor se necesitan los nodos en los cuales se van a ejecutar los trabajos de los usuarios. Los nodos que son configurados para tal fin cumplen el rol que se conoce como Execute Machine. Es importante que un nodo que ejecuta trabajos en el pool de Condor posea buena cantidad de espacio de disco ya que todos los archivos que necesite la tarea al momento de ejecutarse se volcarán al disco local de dicha

máquina. Cabe destacar que puede utilizarse el mecanismo NFS (Network File System), el cual posibilita que distintos sistemas conectados a una misma red accedan a archivos remotos como si se tratara de locales. En este caso, las máquinas locales utilizan menos espacio de disco debido a que los datos se encuentran centralizados en un único lugar pero pueden ser accedidos y modificados por varios usuarios, de tal forma que no es necesario replicar la información.

- Los nodos que ejecutan trabajos cuentan con recursos como ser memoria real, velocidad de la CPU, espacio de swap, etc. Dicho nodo anuncia sus capacidades y estado de uso en el que se encuentra al administrador central, en cual reside el demonio de Condor llamado *Startd*. Cuando el demonio *Negotiator* ha concretado exitosamente la asociación de un nodo con un trabajo, y el nodo desde el cual se emitió el trabajo ha establecido contacto con el demonio *Startd*, este demonio inicializará el demonio *Starter* que es responsable de administrar la ejecución local del trabajo, de establecer el entorno de ejecución y de monitorear el progreso del trabajo una vez comenzada su ejecución.

El demonio *Starter* se comunica a su vez con el demonio *Shadow* el cual se ejecuta en el nodo que se emitió el trabajo. Este demonio retorna información de estado y libera el nodo de ejecución cuando se haya completado el trabajo. Los propietarios de los nodos pueden retomar el control sobre el nodo y como consecuencia de ello se liberarán los trabajos que estén ejecutándose en ese nodo.

- Nodo emisor: desde este nodo se emiten los trabajos de los usuarios, para lo cual se utiliza el comando *condor_submit*. Este comando es un programa que permite enviar trabajos para ser ejecutados en Condor. *Condor_submit* necesita un archivo de emisión del trabajo, donde están contenidos los comandos y requerimientos necesarios para la ejecución del trabajo. Cada vez que se emite una tarea se genera un proceso el cual se debe de mantener y controlar. Por otro lado, en los nodos donde se ejecutan tareas, se genera un proceso en el nodo emisor por cada una de esas tareas. Por esta razón, los nodos emisores de trabajos necesitan una cantidad considerable de memoria real.

También se requiere que el nodo emisor posea bastante espacio de disco duro. Esto es así porque todos los nodos ejecutores de tareas generan unos ficheros llamados “ficheros checkpoint” (este tema se verá en el capítulo 4 de esta tesis) y estos ficheros son almacenados también en el nodo emisor. Estos ficheros suelen ocupar bastante espacio en disco y si se emiten muchas tareas, se necesitará mucho espacio de disco para dar lugar a estos ficheros.

El nodo emisor permite que los usuarios envíen sus trabajos a una cola local virtual. Esta función es controlada por el demonio *Schedd*, el cual para administrar los trabajos encolados, informará al demonio *Collector* respecto del número de trabajos en estado ocioso que tiene en la cola de trabajos. Posteriormente, durante el proceso de negociación, el demonio *Collector* será contactado por el demonio *Negotiator*, el cual reside en el administrador central. Cabe mencionar que el demonio *Collector* es el responsable de recolectar toda la información acerca del pool de Condor,

mientras que el demonio *Negotiator* es el responsable de la asociación de trabajos con los recursos del pool, para lo cual necesita consultar al demonio *Collector* sobre el estado de los recursos del pool.

Luego el planificador emitirá los requerimientos de sus trabajos en orden prioritario de manera de asignar los recursos adecuados. El planificador de trabajos es responsable de administrar de las solicitudes de los trabajos para su ejecución, lo cual implica administrar archivos de entrada, binarios, requerimientos de entorno y otros parámetros, que a su vez pueden pertenecer a múltiples usuarios, y también de mantener una cola persistente de trabajos.

Una vez que se ha logrado la asociación recurso/trabajo, el demonio *Schedd* invocará al demonio *Shadow* que es responsable de la ejecución remota del trabajo. Este demonio interactúa con el demonio *Starter* en el nodo remoto y también monitorea la ejecución del trabajo. Este demonio se encarga de establecer un mecanismo de control llamado checkpointing, realizar llamadas al sistema en el nodo local para la aplicación remota y re-planificar el trabajo en caso de que este falle.

El planificador puede intentar re-utilizar un recurso ya asociado para otro trabajo o reservarlo hasta que termine de utilizarlo el trabajo al cual se asignó.

El demonio *Master* es común a todos los tipos de nodos y se ejecutará en cada nodo del pool de Condor. El demonio *Master* se asegura de que los demonios correspondientes a cada tipo de nodo estén ejecutándose y los reinicia en caso de falla. A través de este demonio se pueden atender de manera remota las solicitudes administrativas como ser inicio y parada o reconfiguración de los demonios.

Mientras que el demonio *Negotiator* en el administrador central puede recomendar la asociación de un nodo con un trabajo, la decisión final de aprobar o no esa asociación queda en manos del nodo, basándose en la configuración que haya realizado su propietario.

Esto lleva a notar la diferencia entre solicitar un nodo para un trabajo y que se logre efectivamente la asociación del trabajo con ese nodo. Una vez que el usuario y el nodo han sido notificados de que la asociación ha sido exitosa, Condor reclama el nodo, el cual luego de que se realice la autenticación mutua basándose en la información brindada por el demonio *Negotiator*, puede ser autorizado o rechazado para utilizar dicho nodo.

Esto le permite a Condor hacer frente al hecho de que el proceso de asociación o matchmaking pueda haber sido realizado basándose en información desactualizada y así poder respetar y proteger al recurso de acuerdo a las políticas de su propietario, las cuales pueden ser distintas a las definidas por el administrador del pool de Condor.

Es importante tener en cuenta el rol de estos componentes cuando, por ejemplo, se está tratando con aplicaciones paralelas y con el paradigma del tipo maestro/esclavo.

Esencialmente este paradigma permite que un proceso central maestro coordine la ejecución de tareas individuales en los procesos esclavos en los nodos remotos, y Condor es en este caso, la plataforma administradora de recursos sobre la que pueden desarrollarse ese tipo de aplicaciones.

Cuando el maestro necesita nodos adicionales con características especificas para ejecutar alguna tarea, los solicitará al administrador de recursos de Condor, el cual asignará un nodo esclavo al cual el maestro emitirá la tarea. En este caso, el demonio *Shadow* brinda el servicio de administrador de recursos para este tipo de aplicaciones, a través del cual los maestros pueden solicitar recursos. A su vez este demonio procederá a emitir estas solicitudes al planificador local.

En la figura 1 se observa la interacción de los demonios anteriormente mencionados:

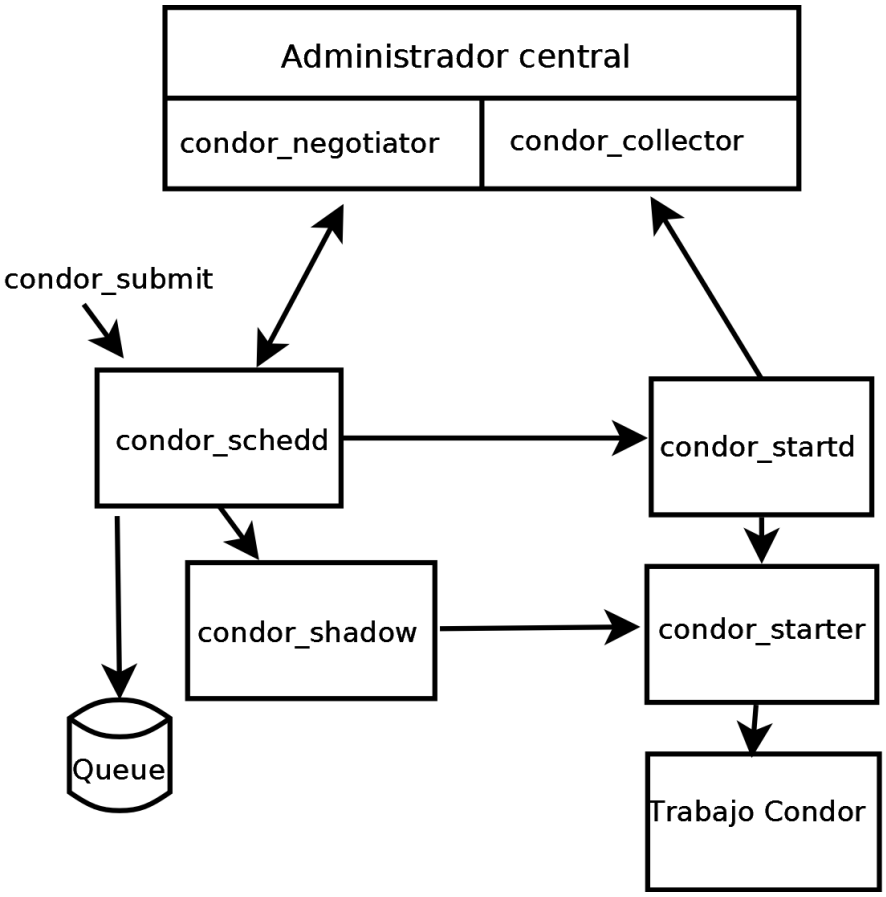

Figura 1. Los demonios de Condor

2.2. Instalación de un pool de Condor

Para realizar las experiencias computacionales de esta tesis, se utilizaron recursos con dedicación parcial, recursos con dedicación total y combinación de ambos tipos de recursos en infraestructuras mixtas.

Los recursos dedicados se disponen en clusters, y los laboratorios de enseñanza proveen de recursos parcialmente dedicados.

Hay dos formas posibles de instalar Condor en un pool de nodos. La primera forma de instalación que se implementó en esta tesis consiste en instalar el middleware nodo por nodo.

Posteriormente se optimizó el proceso de instalación de Condor implementando NFS (Network File System). Este método de instalación ofrece numerosas ventajas, por ejemplo permite acceder a archivos remotos como si fueran locales, se reducen los requerimientos de espacio de disco en las estaciones de trabajo, los archivos están centralizados, lo cual facilita su administración y ayuda a mantener la consistencia de los archivos, entre otras. En la subseción 2.3.2 puede consultarse el proceso de instalación de Condor nodo por nodo y mediante NFS.

2.3 Infraestructura y conectividad utilizada en esta Tesis

En esta sección se presentan las infraestructuras sobre las cuales se han realizado los experimentos computacionales. Las mismas cuentan con recursos dedicados y no dedicados, los cuales son de disponiblidad parcial. En este último caso se dispone de dos laboratorios informáticos pertenecientes a una institución educativa, el Instituto Tecnológico Universitario, sede Redes y Telecomunicaciones. Estos laboratorios de disponibilidad parcial disponen de tiempos ociosos cuando los alumnos de la institución no los utilizan, momentos en los cuales se dispuso de ellos para realizar los experimentos computacionales de esta tesis. Durante el desarrollo de esta Tesis cambió la infraestructura computacional sobre la cual se desarrollaron los experimentos computacionales, ambas infraestructuras se describen en los párrafos siguientes.

Se utilizan además, clusters con recursos computacionales dedicados, los cuales resultan mucho más costosos en comparación a los recursos informáticos de los laboratorios con dedicación parcial. Estos clusters se encuentran en una sala especial con refrigeración, dentro de la misma institución educativa.

Mediante Condor se integraron todos estos recursos distribuidos a fin de que los cálculos con alto costo de gestión puedan ser realizados de forma ágil y eficiente.

En la figura 2 se muestra la infraestructura inicial montada para ejecutar un primer experimento computacional sobre un pool de Condor, el cual fue un trabajo sencillo escrito en lenguaje C. La misma infraestructura se utilizó además para procesar una aplicación de inteligencia artificial para detección de intrusos y una aplicación de mecánica de sólidos. Los resultados de estos experimentos se presentan y discuten en el capítulo 5.

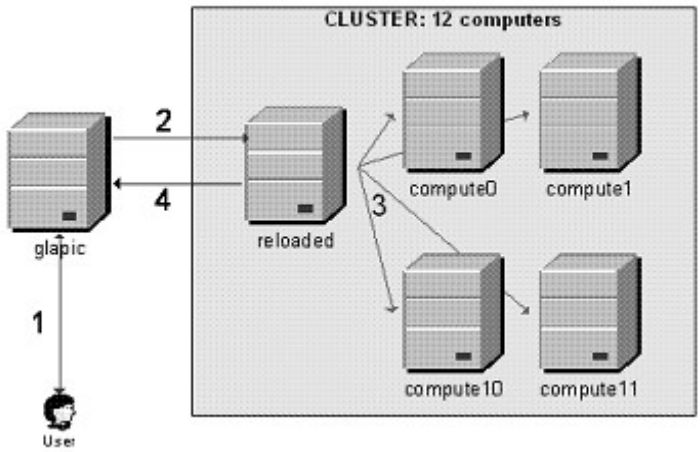

Figura 2. Cluster Reloaded

La computadora llamada Glapic se comporta como un nodo administrador de Condor que distribuye los trabajos a las distintas estaciones del cluster Reloaded. Glapic está compuesta por un procesador P4 de 3.0 GHz, 1 MB de Cache L2, 1 GB de RAM, disco rígido de 160 GB, y un adaptador de red Fast Ethernet.

En Glapic se instaló Fedora Core 3 Kernel 2.6.9, Condor 6.8.4 como sistema operativo y middleware respectivamente. En el cluster Reloaded se instaló la distribución para cluster Rocks 4.1.

El cluster Reloaded tiene una estación de trabajo maestra que actúa como front end y 12 estaciones de trabajo esclavas. Estas últimas están basadas en P4, con procesador de 3.0 GHz, 1 MB de Cache L2, 1 GB de RAM, disco rígido SATA de 80 GB, y un adaptador de red Gigabit Ethernet. La computadora maestra y las estaciones de trabajo del cluster están interconectadas mediante una red Gigabit Ethernet.

Posteriormente, se conformó una infraestructura alternativa para procesar trabajos que requerían mayor poder de cómputo. Para ello, el cluster Reloaded, indicado como Cluster A en la figura 3, se interconectó con dos laboratorios de enseñanza con disponibilidad parcial, a los cuales se accede mediante el equipo denominado Glapic que hace las veces de front end, como se muestra en la figura 3 y con el cluster B, de equipos AMD Opteron. En esta infraestructura se ha procesado una aplicación de inteligencia artificial para detección de intrusos. Los resultados de estos experimentos se presentan en el capítulo 5.

El cluster B que está compuesto por cuatro equipos AMD Opteron Processor 242, de 1.6 GHz y con 2.0 GB de RAM cada uno, donde se instaló Debian Lenny kernel 2.6.18skas y Condor 6.8.4 como sistema operativo y middleware respectivamente y la versión 2 de MPICH. En el anexo B se indican los pasos para instalar MPICH2 nodo por nodo y vía NFS.

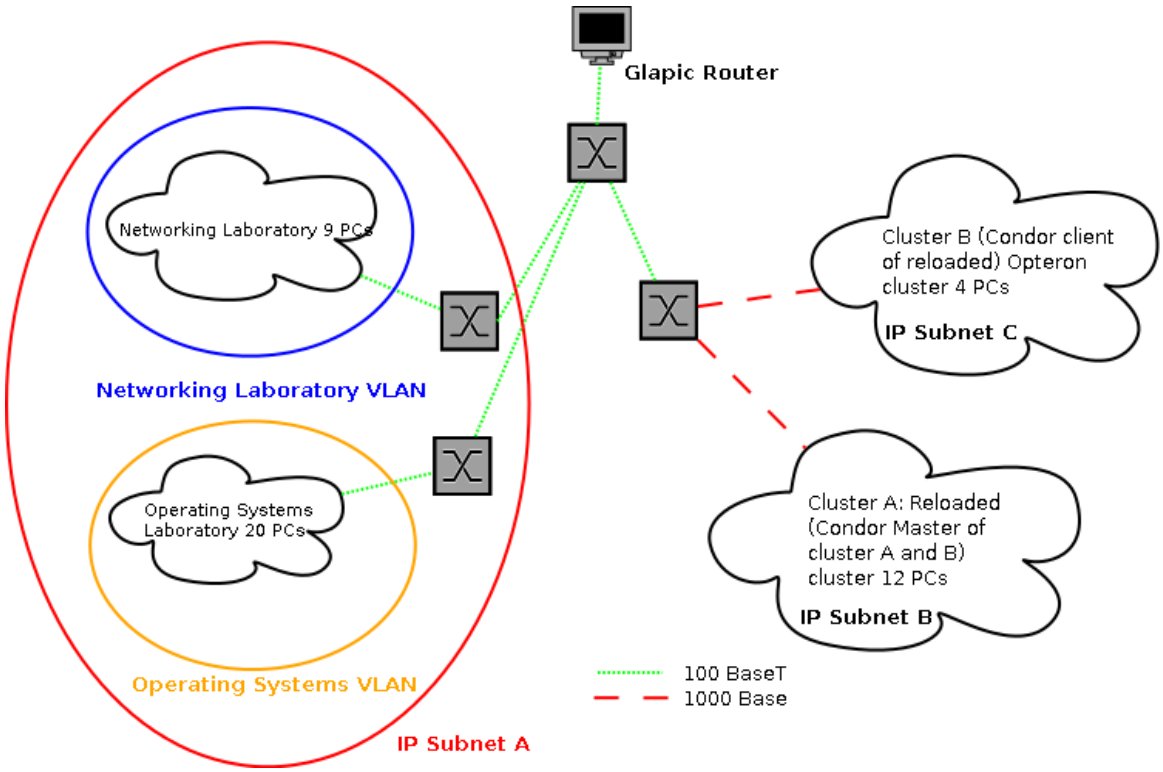

Figura 3. Infraestructura

Los laboratorios de enseñanza, Sistemas Operativos y Redes, cuentan respectivamente con 20 y 9 puestos de trabajo de características similares. Estos equipos poseen procesador Pentium 4 HT de 2.8 GHz, disco rígido de 80 GB, placa aceleradora de video NVidia GForce 4 MX de 128 MB, placa de red FastEthernet Realtek 8139 y lectora de CD Samsung 52X. Cada uno de los equipos de los laboratorios cuenta con 1 GB de memoria RAM.

Posteriormente se adquirió un nuevo cluster, compuesto por procesadores Core 2 Duo de 3.0 Ghz, 4 GB de memoria RAM, 160 GB de disco rígido y adaptador de red Gigabit Ethernet. Este cluster se denomina Twister y consiste en un nodo maestro que actúa como front end y 8 nodos esclavos. En este cluster se instaló la distribución 5.0 de

Rocks como sistema operativo, Condor 7.4.1 como middleware y la versión 2 de MPICH. La nueva infraestructura se muestra en la figura 4.

Cabe destacar que en esta nueva infraestructura, el equipo denominado Glapic en la figura 3 pasó a llamarse Tesla y el cluster denominado Reloaded (cluster A de la figura 3) recibe ahora el nombre de Storm. El hardware, sistema operativo y middleware, han permanecido sin cambio.

En esta nueva infraestructura se procesaron problemas de multiplicación de matrices y un modelo meteorológico. Los resultados de estos experimentos se presentan en el capítulo 5.

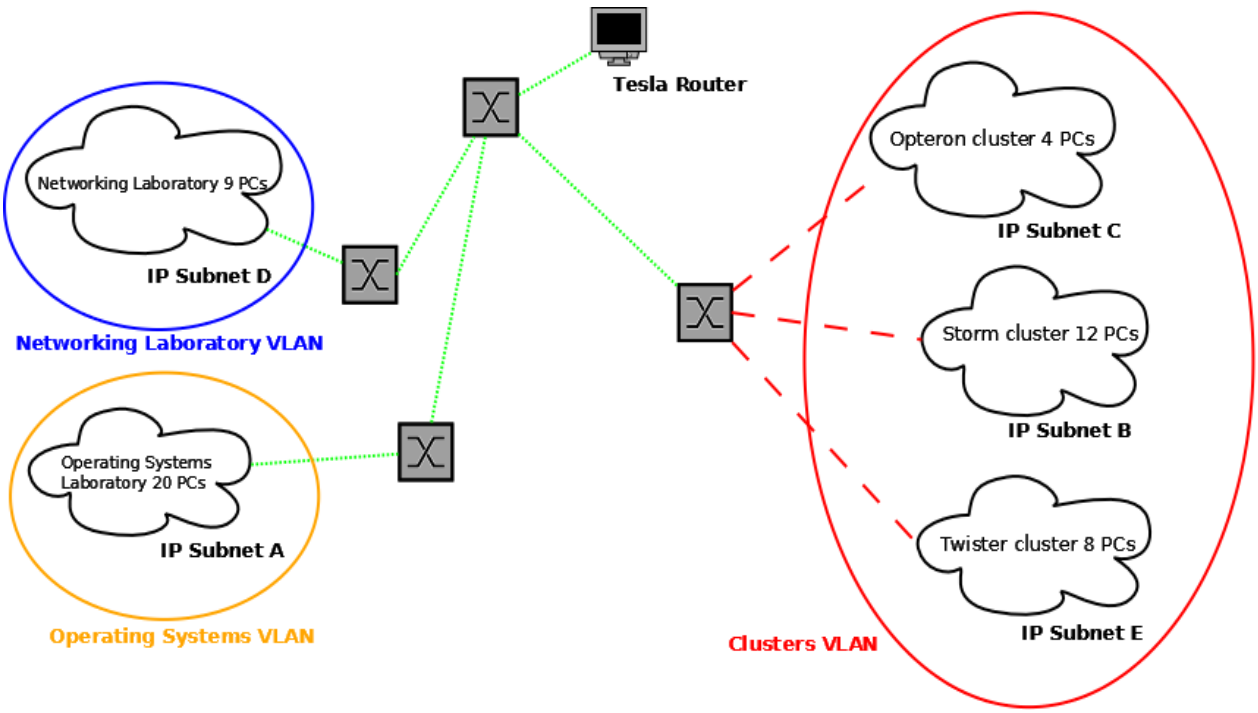

Figura 4. Infraestructura alternativa

2.3.1 Conectividad

En la infraestructura alternativa de la figura 4 se observa que el laboratorio de Sistemas Operativos cuyos equipos conforman la subred IP A, con un rango de direcciones IP desde 192.168.218.1 hasta 192.168.218.20 con máscara 255.255.255.224. Los equipos del laboratorio de Redes conforman la subred IP D que abarca las direcciones IP desde 192.168.218.33 hasta 192.168.218.41 con máscara 255.255.255.224. El cluster A

conforma la subred IP B que abarca las direcciones IP desde 10.255.255.243 hasta 10.255.255.254, con máscara 255.255.0.0. El cluster B conforma la subred IP C que abarca las direcciones IP desde 10.2.2.2 hasta 10.2.2.5 con máscara 255.255.0.0. Los equipos del cluster A y B están interconectados por una red Gigabit Ethernet.

El cluster de 8 nodos, denominado Twister conforma la subred IP E que abarca las direcciones IP desde 10.10.10.1 hasta 10.10.10.8 con máscara 255.255.255.0.

Tesla es un equipo con múltiples interfaces de red, cada una con una dirección IP, a través de las cuales mantiene comunicación con cada uno de los laboratorios de enseñanza y con el nodo front end de Storm. A su vez, este nodo también tiene configuradas múltiples interfaces de red, cada una con una dirección IP para comunicarse con los cluster A y B y con Tesla. En Tesla se han configurado rutas estáticas hacia cada una de las subredes de los cluster A y B a fin de lograr comunicación con los nodos respectivos y a su vez el nodo front end de Storm tiene rutas estáticas hacia los laboratorios de enseñanza, a los cuales llega mediante Tesla.

Por razones de administración de las redes de la institución educativa se han dispuesto sendas VLANs (Redes de Área Local Virtual) en los laboratorios de Sistemas Operativos y Redes respectivamente configuradas mediante diferentes conmutadores o switches Fast Ethernet. El puerto del switch al que está conectado Glapic pertenece a ambas VLANs para mantener la conectividad con ambos laboratorios. Una VLAN es una agrupación lógica de dispositivos y/o usuarios, los cuales se pueden agrupar por función, departamento, aplicación, etc, sin importar su ubicación geográfica. Las VLANs tienen la característica de acotar el dominio de difusión, y si es necesario asociar dos redes separadas por VLANs es necesario recurrir a un dispositivo de enrutamiento de capa tres.

2.3.2 Instalación del middleware Condor

Instalación nodo por nodo

Los nodos de los pool de Condor utilizados en los experimentos computacionales utilizan a Linux/GNU como sistema operativo. En estos nodos se ha instalado el paquete rpm para Condor, de acuerdo a los siguientes pasos. Puede consultarse información más detallada del proceso de instalación en el manual de Condor disponible en línea [15]:

 # rpm -i condor-7.4.1-linux-x86-rhel5-1.i386.rpm

Se añade el siguiente script en el archivo /etc/profile. Este archivo se ejecuta cada vez que un usuario inicia una sesión.

 # . /opt/condor-7.4.1/condor.sh

Se agregan permisos de ejecución para todos los usuarios al script condor.sh:

 # chmod a+x condor.sh

Rpm es un administrador de paquetes que puede utilizarse para instalar, consultar, verificar, actualizar y borrar paquetes de software. Un paquete de software consiste en un archivo de ficheros y meta-datos utilizados para instalar y borrar el archivo de ficheros. Los meta-datos incluyen scripts de ayuda, atributos del fichero e información descriptiva acerca del paquete.

Luego se configura el archivo /etc/hosts con los nombres de todos los hosts del pool. Se editan los archivos de configuración de Condor de acuerdo a lo indicado en el manual [16]. A continuación se detallan algunas variables que se deben modificar en una configuración típica de Condor.

- CONDOR_HOST = maestro

  En esta variable se define el nombre cmpleto o dirección IP del maestro del pool de Condor.

- DAEMON_LIST = COLLECTOR, NEGOTIATOR, MASTER, SCHEDD, STARTD

  Esta variable lista los demonios que se ejecutarán en un nodo maestro de Condor. Un nodo de ejecución de Condor de trabajos necesita que estén activos los demonios MASTER Y STARTD.

- BIND_ALL_INTERFACES = true

  Esta variable permite que todos los demonios de Cóndor escuchen en todas las interfaces de red disponibles. Cabe destacar que cuando un nodo maestro sólo recibe peticiones de una única red privada, no es imprescindible que esta variable esté en true. Sin embargo, cuando el maestro de un pool de Condor puede recibir peticiones de más de una red privada, es necesario que esta variable esté configurada con el valor true.

Instalación del middleware Condor vía NFS

En caso de ser posible, es recomendable la instalación del middleware mediante NFS, ya que este método ofrece las siguientes ventajas:

- NFS permite a los nodos remotos montar un sistema de archivos sobre una red e interactuar con esos sistemas de archivos como si estuvieran montados localmente. Esto permite a los administradores de la red consolidar los recursos en servidores centralizados en la red.

- No hay necesidad de duplicar datos en la red, ya que los mismos pueden ser accedidos desde cualquier nodo en la red a una única locación.
- Los directorios home, propios de cada usuario, pueden almacenarse en un único lugar de la red y ser accedidos por los usuarios a través de la red.
- A nivel de seguridad, tiene el beneficio de que hay un solo sistema de archivos donde implementar políticas de seguridad y de backup.

Configuración del servidor desde el que se hará la instalación

Se instalan los paquetes rpm para Condor en el servidor:

 # rpm -i condor-7.4.1-linux-x86-rhel5-1.i386.rpm

 # mkdir /opt/condor_local-7.4.1

Se edita el archivo de Condor llamado condor_config para que apunte al directorio local y al archivo de configuración local condor_config_local:

LOCAL_DIR = /opt/condor_local-7.4.1/

LOCAL_CONFIG_FILE = /opt/condor_local-7.4.1/condor_config.local

Se edita el archivo /etc/exports. Se especifica la dirección IP de los recursos que deben tener acceso al recurso compartido:

/opt/condor-7.4.1 10.1.1.0/255.0.0.0(rw,async)

Se reinicia el servicio NFS:

 # service nfs restart

Configuración de los nodos del pool de Condor.

Se edita el archivo /etc/fstab de cada nodo del pool, siendo Storm del maestro del cluster. El archivo /etc/fstab es referenciado por el servicio NFS al momento del arranque.

storm:/opt/condor-7.4.1/ /opt/condor-7.4.1/ nfs defaults 0 0

En este caso, storm es el nombre del servidor que exporta el sistema de archivos, /opt/condor-7.4.1/ es la ruta al directorio exportado, /opt/condor-7.4.1/ indica el sistema de archivos local en el cual se montará el directorio exportado. Cabe destacar que este punto de montaje debe existir antes de que /etc/fstab sea leído o el montaje fallará.

Se crean los directorios condor-7.4.1 y condor_local-7.4.1:

 # mkdir /opt/condor-7.4.1

 # mkdir /opt/condor_local-7.4.1

 # mount -a

 # scp -r storm:/opt/condor_local-7.4.1/ /opt/condor_local-7.4.1/*

 # vi /etc/profile

source /opt/condor-7.4.1/condor.sh

 # . /etc/profile

 # condor_master

2.3.3 Agrupamiento bajo Condor

Una vez que se han resuelto los requisitos de conectividad a nivel de la capa de red, señalados en la subsección anterior y que Condor no puede resolver de manera autónoma, pueden ejecutarse trabajos utilizando recursos del cluster (Subredes IP B y C) y de los laboratorios (Subredes IP A y D). Es importante mencionar que para garantizar la conectividad (a nivel IP) necesaria para que Condor pueda emplear los recursos no dedicados, es necesario contar con la colaboración y participación del administrador de la red.

Tesla puede administrar conjuntamente al hardware de ambos laboratorios de enseñanza, los cuales, para Condor conforman un único conjunto o pool de 29 equipos. Como se ha mencionado, tanto Tesla como el front end del cluster Storm cuentan con múltiples interfaces de red. En estos casos Condor tiene la facilidad de asociarse y escuchar en todas ellas mediante la configuración de la variable BIND_ALL_INTERFACES = True del archivo de configuración local de Condor. Esta posibilidad permite a Condor recolectar desde el nodo Tesla, la información del estado de los recursos a través de las interfaces correspondientes a las distintas redes.

Si se desea mantener independencia de los laboratorios y/o clusters, entonces los mismos pueden agruparse en diferentes pool. En este caso también es posible utilizar los recursos de manera conjunta, para lo cual se toma ventaja de la característica de flocking, propia de la administración de pool de Condor y que se discute en la subsección 2.4.1.

Es importante destacar que para el caso de la infraestructura no dedicada, el propietario del recurso tiene prioridad absoluta. Si el recurso está ocioso, puede ser utilizado por cualquier usuario y para volver a la actividad no es necesaria una acción en particular.

2.4 Condor y Grid

El objetivo de la computación grid es permitir la utilización de recursos que están distribuidos a lo largo de distintos dominios administrativos, los cuales a su vez pueden pertenecer a distintos dueños. Cabe destacar que resultaría poco factible combinar todos estos recursos en un gran pool de Condor, ya que su administración resultaría más compleja así como también la coordinación entre los distintos administradores.

Para resolver estos aspectos, Condor aporta mecanismos nativos para la computación grid y para la interconexión con sistemas grid.

En esta sección se describen los mecanismos de flocking, Condor-G, Condor glidein, Condor-C y Globus y se discute el mecanismo elegido para ejecutar los experimentos computacionales.

2.4.1 Condor Flocking

Para permitir que múltiples pool de Condor compartan recursos y se puedan enviar trabajos desde un pool a otro, Condor implementa el mecanismo de flocking, con el cual se realizaron experimentos en esta tesis. Un pool de Condor puede configurarse para que acepte solicitudes de ejecución de trabajos de otro pool remoto. El mecanismo de flocking funciona de la siguiente manera:

Si un pool A desea permitir que los usuarios de otro pool B ejecuten sus trabajos sobre sus recursos, el administrador central del pool A debe configurarse para permitir tal alternativa. Además, los demonios de Condor que se están ejecutando en el nodo B (emisor del trabajo) se configuran especialmente para utilizar los recursos disponibles

en el pool A. Los nodos del pool B sólo enviarán trabajos al pool A si los recursos locales no están disponibles o están en uso.

Por otro lado, puede configurarse el archivo de emisión del trabajo para que los mismos se ejecuten en el pool que el administrador de Condor decida, ya sea por preferencias de hardware, sistema operativo, etc.

La negociación para la planificación del trabajo ocurre entre el nodo que emitió el trabajo (en este caso un nodo del pool B) y el administrador central del pool A, y luego el trabajo se ejecutará en el recurso remoto. Es importante destacar que ambos pool deben estar configurados para compartir recursos [17], como se muestra en las figuras 5 y 6.

En el caso de no disponer de recursos suficientes para satisfacer los requerimientos de un trabajo, Condor puede disponer de los recursos de otro pool del que tenga conocimiento. Así los trabajos migran de un pool a otro dependiendo de la disponibilidad de recursos para ejecutar trabajos. Siempre se intenta que los trabajos se ejecuten en el pool desde el que se emitió, solo migrarán a otro pool cuando los recursos locales no sean suficientes.

En la infraestructura disponible en esta tesis para ejecutar los experimentos computacionales, Tesla actúa como maestro Condor de los laboratorios de enseñanza, que conforman un pool de recursos con disponibilidad parcial y Storm actúa como maestro de los cluster A y B, conformando un pool de equipos con recursos dedicados (figura 4).

Los trabajos se emiten desde el administrador central Tesla, el cual compara los recursos ofrecidos por los laboratorios de enseñanza con los de la petición del usuario, es decir realiza el matchmaking. Si los trabajos no pueden ejecutarse en los laboratorios de enseñanza porque los recursos no son suficientes, Tesla utilizará los recursos de los cluster A y/o B. Esta característica se conoce en la jerga Condor como flocking. Para hacer uso de esta facilidad de Condor se configuran dos variables FLOCK_TO y FLOCK_FROM en el archivo de configuración global de Condor.

En la variable FLOCK_TO se indican, separados por coma, los administradores centrales de los pool a los que podrán enviarse trabajos para ejecutar, en caso de no contar con los recursos necesarios para procesar los trabajos En la variable FLOCK_FROM se especifican los recursos del pool a los que se permite el acceso mediante el mecanismo de flocking para procesar trabajos en caso de no contar con los recursos necesarios.

En la figura 5 se muestra parte del archivo de configuración global del administrador central de los laboratorios de enseñanza (Tesla) y en la figura 6 se muestra parte del archivo de configuración global del maestro Condor de los cluster A y B (Storm).

Otras dos variables a considerar son HOSTALLOW_WRITE y HOSTALLOW_READ que permiten respectivamente, definir los recursos que pueden unirse al pool y los recursos que pueden ver el estado del pool pero no pueden unirse al mismo.

```
FLOCK_FROM =  storm.itic.uncu.edu.ar
FLOCK_TO =  storm.itic.uncu.edu.ar
HOSTALLOW_WRITE = *.uncu.edu.ar
HOSTALLOW_READ = *
```

Figura 5. Archivo de configuración de Tesla

```
FLOCK_FROM = tesla.itic.uncu.edu.ar
FLOCK_TO = tesla.itic.uncu.edu.ar
HOSTALLOW_WRITE = *.uncu.edu.ar
HOSTALLOW_READ = *
```

Figura 6. Archivo de configuración de Storm

En el capítulo 5 se presenta y discute un experimento computacional que hace uso de la funcionalidad de flocking.

2.4.2 Condor-G

Condor-G [18] es el nombre que recibe Condor cuando los trabajos del universo grid son enviados a a recursos grid utilizando el software Globus [19] para su ejecución. Un universo en Condor, define un entorno de ejecución, y el universo bajo el cual se ejecutará un trabajo se especifica en el archivo de emisión del trabajo.

Condor-G permite enviar trabajos a una cola y llevar un registro del ciclo de vida de los mismos, administrar la entrada y salida de archivos, etc. Condor-G toma su nombre por la forma en la que se comunica con la parte de Condor que administra recursos. En lugar de utilizar los protocolos de Condor desarrollados para iniciar la ejecución de trabajos en un nodo remoto, Condor-G utiliza al Toolkit de Globus para iniciar el trabajo en el nodo remoto.

En 1998, comenzaron los primeros trabajos sobre computación Grid y uno de los primeros elementos que se desarrolló fue una interfaz uniforme para ejecución batch. El Proyecto Globus diseñó el protocolo GRAM [18] para cubrir esta necesidad. GRAM ofrece una abstracción para ejecución y encolamiento de procesos remotos y permite implementar mecanismos de seguridad fuertes y transferencia de archivos. El Proyecto

Globus ofrece un servidor que trabaja con el protocolo GRAM y convierte a sus comandos a un formato que entiende la mayoría de los sistemas batch.

Para utilizar GRAM, un usuario debe contar con un sistema que recuerde los trabajos que se han enviado, dónde han sido enviados y lo que estén ejecutando. En el caso de que el trabajo falle, el sistema debe analizar la falla y volver a enviar el trabajo, en caso de ser necesario.

Para realizar un seguimiento de un gran número de trabajos, los usuarios necesitan encolamiento, establecimiento de prioridades, funciones de registro, etc. Para brindar este servicio, Condor adaptó el agente estándar Condor-G para comunicarse con GRAM, como se muestra en la figura 7.

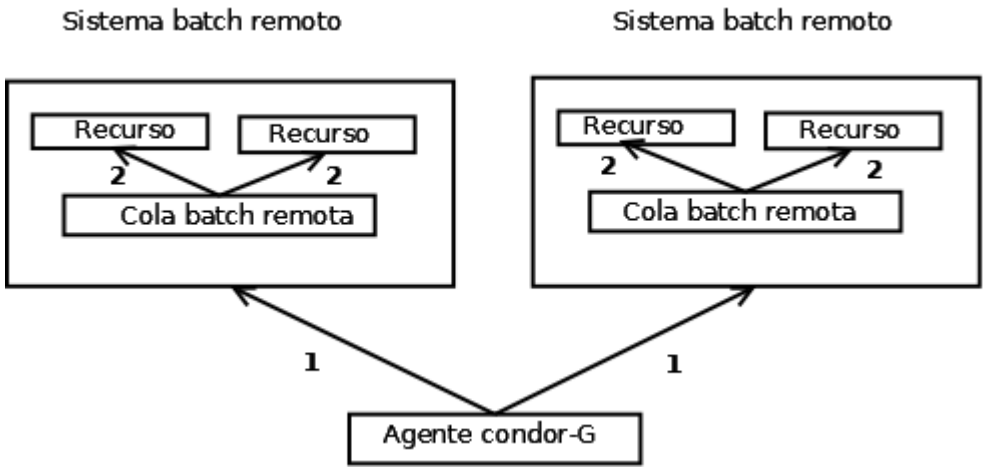

Figura 7. Condor-G: un agente Condor-G ejecutando dos trabajos de forma remota.

En la figura 7 se observa:

1. El agente Condor-G ejecuta dos trabajos a través de colas batch remotas.
2. Los trabajos esperan por recursos ociosos para comenzar a ejecutarse.

Una desventaja de Condor-G es que no soporta todas las características de cada sistema batch existente por debajo de GRAM, por lo que si GRAM incluye todas las características de cada sistema subyacente, resultaría complejo e inutilizable.

Condor-G se apoya en avances recientes en dos áreas distintas: (1) la seguridad y el acceso a los recursos en ambientes de dominios múltiples, como se es soportado dentro de las herramientas de Globus, y (2) la administración de la capacidad de cómputo y el agrupamiento de recursos dentro de un dominio administrativo único, incorporado dentro del sistema de Condor.

Condor-G combina los protocolos de manejo de recursos entre dominios de Globus y los métodos de manejo de recursos y envío de trabajo dentro del dominio de Condor para permitirle al usuario agrupar recursos multi-dominios como si pertenecieran a un dominio personal.

Condor-G provee a la comunidad de cómputo grid de un poderoso ejecutor de tareas con muchas características. Usado como entrada para un grid computacional, Condor-G puede manejar miles de trabajos destinados a ser ejecutados en sitios distribuidos. Provee monitoreo de trabajos, registro, notificación, tolerancia a fallos, manejo de credenciales y puede manejar complejas interdependencias. Los comandos de Condor-G son apropiados para ser usados directamente por usuarios finales, o para ser usados en interfaces con ejecutores de trabajos de más alto nivel y portales Web.

En el anexo A se indican los pasos para instalar Globus y se muestra la interacción de Condor-G con Globus mediante un experimento computacional. Es importante mencionar que se requieren conocimientos de administración de Linux para poder llevar a cabo la instalación de Globus, aunque el manual de Globus y los errores que se reportan a la hora de la instalación son claros y relativamente rápidos de resolver.

2.4.3 Condor Glidein

Glidein [20] es un mecanismo por el cual uno o más recursos grid remotos se unen temporalmente a un pool local de Condor. El programa *condor_glidein* se utiliza para agregar un nodo al pool del middleware, y este recurso estará disponible para su uso mientras sea parte del pool local. Por defecto, este recurso estará disponible para ser utilizado sólo por el usuario que lo agregó al pool, pero será visible para todos los usuarios.

Condor_glidein instala y ejecuta los demonios de Condor necesarios en el recurso remoto, de tal forma que este recurso pueda reportarse y unirse al pool local.

Una vez que el recurso remoto está correctamente configurado, comienza la tarea de ejecución que inicia los demonios del middleware. Los trabajos que se ejecutan en el recurso remoto deben pertenecer al universo grid.

Un caso de uso del mecanismo de glidein es cuando un usuario desea ejecutar trabajos en el universo Estándar sobre recursos administrados por Globus, para así contar con la planificación de trabajos, checkpointing, migración y llamadas remotas al sistema.

El usuario puede utilizar el universo Globus para ejecutar demonios de Condor sobre recursos Globus y cuando los recursos ejecutan estos trabajos “glidein”, se unirán temporalmente al pool de Condor del usuario. De esta manera, el usuario podrá enviar trabajos de los universos Estándar, Vanilla o Parallel, los cuales serán asociados y ejecutados sobre los recursos Globus.

El mecanismo de Glidein presenta algunas consideraciones particulares a tener en cuenta

- Para utilizar Glidein es necesario configurar las variables HOSTALLOW_WRITE y GLIDEIN_SITES. Una vez que estas variables estén

configuradas, será posible asignar dinámicamente recursos adicionales.

- El sistema es muy sensible a la presencia de firewalls, y necesitará varios puertos adicionales abiertos para que puedan comunicarse los demonios de Condor. Es importante mencionar el costo que implica la administración de un firewall y la imposibilidad de modificar su configuración cuando no esta a cargo de uno.

2.4.4 Condor-C

Condor-C es una forma de conectar múltiples recursos sin necesidad de acoplarlos a nivel del sistema. El objetivo de Condor-C es permitir el acceso a pool remotos del middleware Condor sin implementar el mecanismo de flocking entre ellos. Condor-C permite que los trabajos de una cola de un nodo se muevan a la cola de trabajos de otro nodo. Condor-C permite que desde la máquina desde la cual se emiten los trabajos mantenga un único proceso y una única conexión de red con la máquina remota, sin importar el número de trabajos encolados o en ejecución.

Condor-C tiene algunas limitaciones, por ejemplo la ejecución de trabajos en los universos Estándar, Java y Parallel no han sido probados, así como tampoco los métodos de autenticación GSI y Kerberos.

2.4.5 Condor y Globus

El conjunto de herramientas Globus es fundamental para habilitar la tecnología necesaria para conformar una grid para que los usuarios compartan poder de cómputo, bases de datos, y otras herramientas de manera segura traspasando fronteras corporativas, institucionales y geográficas sin sacrificar la autonomía local. Este conjunto de herramientas incluye programas y librerías para el monitoreo, descubrimiento y administración de recursos además de la seguridad y manejo de archivos.

- GT4 (Globus Toolkit versión 4) cumple con los últimos estándares de servicios Web de la Organización para la Interoperabilidad de Servicios Web (WS-I), los cuales proveen la máxima interoperabilidad entre ambientes diferentes.
- GT4 incluye soporte inicial para estándares importantes de autorización, incluyendo el Lenguaje de Marcado de Seguridad (SAML) y el Lenguaje de Marcado de Control de Acceso Extensible (XACML); esto provee una base para una habilitar infraestructura de grid de servicios Web seguros.
- GT4 implementa el Marco de Recursos de Servicios Web (WS-RF) y las especificaciones de Servicios de Notificación de Servicios Web (WS-N), los cuales son estándares emergentes en OASIS respaldados por la mayoría de los vendedores de servicios Web habilitados para grid y sistemas de manejo de

recursos.

- Las características de autorización y capacidades de seguridad del GT4 hacen del conjunto de herramientas listo para empresas desde una perspectiva de seguridad.
- Hay 4 componentes principales de Globus:
    - Seguridad (GSI)
    - Manejo de datos (GridFTP, RFT)
    - Manejo de recursos (GRAM)
    - Servicios de información (Index Services)

Por otro lado, la meta del proyecto Condor es desarrollar, implementar, desplegar, y evaluar mecanismos y políticas que soporten Computación de Alto Rendimiento (HTC) en grandes grupos de recursos de computación distribuidos.

La computación grid y la capacidad de las comunidades de compartir recursos han emergido como una importante faceta de la computación. Condor-G es el producto de la unión de tecnologías provenientes de los proyectos Condor y Globus.

Condor-G es usado como entrada para un grid computacional y con respecto a Globus, Condor-G añade confiabilidad a la hora de emitir trabajos, y le permite al usuario emitir los trabajos a colas de trabajos, llevar un registro detallado del ciclo de vida de los trabajos, administrar archivos de entrada y salida, etc.

Condor-G presenta diferencias con Globus:

- Permite enviar varios trabajos de una sola vez y luego monitorearlos.
- Los usuarios pueden recibir notificaciones cuando los trabajos terminan normalmente o fallan.
- Condor-G es tolerante a fallas del sistema, ya que pueden recuperarse las funcionalidades de Condor-G una vez que el sistema se recupera.
- Condor-G administra tanto colas de trabajos como de recursos de uno o más sitios donde se ejecutan los trabajos.
- Condor-G se comunica con los recursos y transfiere archivos desde y hacia los recursos utilizando mecanismos de Globus, como GSI, protocolo GRAM para emitir trabajos y al servidor GASS local para transferencia de archivos.

2.4.6 Comparación y selección de un mecanismo para conformación de la grid

Tanto los mecanismos de flocking como glidein permiten la ejecución de trabajos en otros universos aparte del universo Grid. En cambio, en el caso de Condor-C, depende del planificador remoto.

Excepto en el caso de flocking y glidein, es necesario indicar que los trabajos serán enviados a un planificador remoto, no pudiéndose ejecutar de manera local aunque hubieran recursos disponibles. Como consecuencia de esto puede ocurrir que haya trabajos en cola de espera para ser ejecutados en los recursos remotos habiendo recursos locales ociosos.

Una desventaja de los métodos de flocking y glidein es que requieren una conexión permanente de red ya que las interrupciones en la red pueden ocasionar que los trabajos no terminen su ejecución. En el caso de los otros métodos, al producirse una desconexión habiéndose ya enviado el trabajo al planificador remoto, éste podrá continuar ejecutándose normalmente.

Otra consideración importante de los métodos de flocking y glidein es que el planificador y el nodo de ejecución de los trabajos deben poder establecer comunicación a nivel de red de forma bidireccional. Para los restantes métodos , sólo habrá que tener en cuenta que el planificador remoto pueda comunicarse con el nodo de ejecución. El problema surge cuando el nodo de ejecución se encuentra en una red privada. Es importante mencionar que si hay un firewall en el medio de estos dos nodos, habrá que configurarlo para que permita el establecimiento de la comunicación entre ellos, abriendo los puertos necesarios. Además hay que realizar las configuraciones correspondientes de NAT (Network Address Translation). Este método traduce las direcciones IP privadas de la red en una IP pública para que la red pueda enviar paquetes al exterior; y luego traduce nuevamente esa IP pública a la IP privada de la máquina que envió el paquete, para que pueda recibirlo una vez llega la respuesta.

El mecanismo de flocking requiere que cualquier recurso remoto al cual se acceda esté administrado por Condor, lo cual no siempre es necesario para los demás métodos.

En el mecanismo de glidein, el planificador local delega temporalmente la responsabilidad a otro planificador. Además el mecanismo de glidein necesita de Condor-G para iniciar los demonios de ejecución de Condor en el recurso remoto.

Se ha preferido implementar el mecanismo de flocking porque permite de una manera sencilla hacer uso de recursos de otro pool cuando los del pool local no son suficientes y porque el planificador de Condor local es el responsable directo de asignar y negociar con los recursos que ejecutarán el trabajo. Cabe destacar que en la infraestructura computacional sobre la cual se realizaron los experimentos computacionales con el mecanismo de flocking (figura 4) no cuenta con la presencia de un firewall y la comunicación entre el planificador y los nodos de ejecución es bidireccional.

Otro motivo de esta elección es la facilidad de configuración del mecanismo de flocking, ya que sólo hace falta configurar dos variables en el archivo de configuración global, como se indicó en la subsección 2.4.1. En el capítulo 5 se presenta y discute un experimento computacional que hace uso de este mecanismo. Cabe destacar que el mecanismo de flocking por sí solo no ofrece seguridad a la hora de conformar una grid, aspecto que sí resuelve Globus, donde las comunicaciones están encriptadas. Una alternativa que se propone en esta tesis es comunicar dos pools de Condor mediante flocking a través de una VPN, como se muestra en la sección 3.16 del capítulo 3.

Capítulo 3

3. Seguridad en un pool de Condor

En este capítulo se presentan los mecanismos de seguridad que se implementan en un entorno Condor y como aporte de esta tesis se discute la implementación de una Red Privada Virtual como una forma de comunicar dos pool de Condor a través de un firewall.

Los aspectos de seguridad que deben considerarse en una instalación del middleware son bastante amplios. Debido a que el objetivo principal de Condor es que los usuarios puedan ejecutar código en un gran número de recursos, resulta importante limitar el acceso al pool y definir los privilegios que tendrán estos usuarios a la hora de utilizar el pool. En este capítulo se describen las consideraciones y alternativas posibles para segurizar un entorno computacional de Condor.

3.1 El modelo de seguridad de Condor

Las comunicaciones en el interior de Condor se someten a varios controles de seguridad, por ejemplo una solicitud de un demonio a otro puede requerir autenticación para evitar que el sistema corra algún tipo de riesgo.

Para lograr esto se hace uso de varios protocolos de seguridad para las comunicaciones entre dominios y para estandarizar el acceso a distintos sistemas batch remotos.

Las características de seguridad del middleware son soportadas mediante una librería de comunicación de pasaje de mensajes que han desarrollado los autores de Condor, llamada CEDAR [21]. Esta librería permite que se envíen comandos a través de TCP y/o UDP y brinda funcionalidades de autenticación, autorización y encriptación. En la figura 8 se observa el modelo de seguridad de Condor.

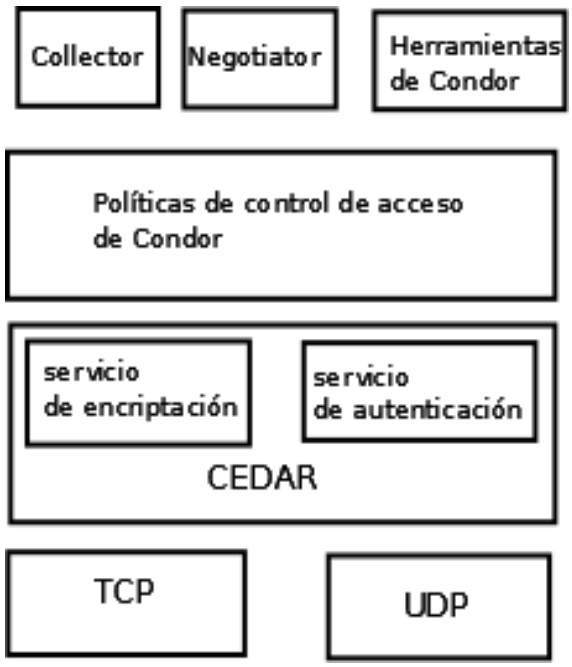

Figura 8. El modelo de seguridad de Condor

Las solicitudes a Condor están categorizadas en grupos de niveles de acceso, basándose en el tipo de operación solicitada. El usuario de una solicitud debe ser autorizado de acuerdo al nivel de acceso solicitado. Por ejemplo la ejecución del comando *condor_status* necesita permisos de lectura. Otras acciones que implican tareas de administración, como reinicio o eliminación de demonios requiere permisos de administrador.

En Condor, la comunicación y/o invocación de comandos se compone de dos partes, un lado se identifica como el cliente y el otro lado se identifica como el demonio. El cliente es la parte que inicia el comando, y el demonio es la parte que procesa el comando y devuelve una respuesta.

En algunos casos es sencillo distinguir el cliente del demonio, pero en otros, no tanto. Los comandos c*ondor_submit* y *condor_config_val* son clientes que envían comandos a los demonios y actúan como clientes durante toda la comunicación. Por ejemplo, el comando *condor_submit* se comunica con el comando *condor_schedd* y en este caso, el demonio que inició el comando cumplirá el papel de cliente. Por ejemplo, el demonio *condor_negotiator* actúa como cliente cuando se contacta con el demonio *condor_schedd* para iniciar el proceso de matchmaking. Una vez que se ha realizado la asociación de un trabajo a un recurso, el demonio *condor_schedd* actúa como cliente y contacta a su vez, al demonio *condor_startd*.

Los comandos en Condor se ejecutan sobre redes TCP/IP y mientras que las comunicaciones de red permiten al middleware administrar los recursos que se distribuyen en una organización o más allá de la misma, también representan algunos riesgos de seguridad. Condor debe asegurarse que los comandos son ejecutados por

usuarios de confianza, los trabajos que contienen datos sensibles deben estar encriptados de tal forma que no sean vistos desde afuera y que no sean modificados. Estos aspectos se cubren con los mecanismos de autenticación, encriptación e integridad que ofrece Condor [22].

3.2 La seguridad en Condor

Los usuarios de Condor deben poder ejecutar código arbitrario, por lo que es importante limitar el acceso al pool y definir los privilegios sobre los recursos del mismo. Es importante destacar que el middleware no puede evitar ataques al sistema provocados por usuarios que hayan elevado sus privilegios a administrador. Sin embargo, Condor puede asegurar que sólo los usuarios de confianza puedan acceder al pool. Además, Condor puede encriptar los datos y verificar la integridad de los mismos [22].

Los aspectos de seguridad más importantes en Condor son:

Usuarios: es preferible que los demonios de Condor se ejecuten bajo el usuario root, mientras que otras tareas comunes sean ejecutadas por usuarios comunes, por ejemplo por el usuario condor.

Autenticación: por defecto Condor determina la identidad de los usuarios mediante un identificador de usuario (user id o uid), pero puede utilizar también mecanismos de autenticación más sofisticados como Kerberos y GSI.

Autorización: una vez que Condor autentica al usuario, define qué tiene permitido realizar en el sistema, pudiendo algunos de ellos sólo enviar trabajos para su ejecución y otros tener privilegios de administrador.

Privacidad: Condor puede encriptar datos provenientes de una comunicación interna. Estos datos pueden ser archivos comunes o ejecutables. Es importante destacar que los datos se encriptan solo para su transmisión en la red, luego son almacenados sin encriptar en el disco.

Integridad: Condor puede enviar datos criptográficos adicionales para verificar que la red de transmisión de datos no ha sido afectada. De esta manera pueden evitarse ataques del tipo “man in the middle”, en el cual el atacante cambia los datos enviados por la red de datos sin que ninguno de los extremos, transmisor y/o receptor lo noten. La verificación de la integridad de los datos solo se realiza para datos que se transmiten por la red, no para los datos almacenados en el disco.

3.3 Cuentas de usuarios en Condor

En los sistemas operativos Linux, la identificación de los usuarios (uid) forma parte de las herramientas del sistema operativo para mantener un control en el acceso de los usuarios. Cada programa que se ejecuta tiene un identificador (uid), un identificador único que identifica a cada usuario que ejecuta un programa. Este identificador también se llama uid real. Puede suceder que un usuario ejecute un programa que es propiedad

de otro usuario, de hecho muchos comandos del sistema funcionan así, por ejemplo un usuario puede ejecutar un programa que pertenece a root. Dado que el programa puede requerir privilegios de root, privilegios que el usuario común no tiene, se configura un bit especial en la especificación de seguridad del programa a fin de permitir que sea ejecutado con el uid del propietario del programa en lugar de que sea ejecutado con el id del usuario del programa. Este uid se denomina uid efectivo.

Condor precisa que los demonios se ejecuten como root, así podrán cambiar el uid efectivo en caso de ser necesario. Cuando los demonios se ejecutan como root, el uid efectivo y el identificador de grupo gid corresponden al usuario y grupo condor.

Cabe destacar que en los experimentos computacionales de esta tesis los demonios se han ejecutado en todos los casos como root y los trabajos se han lanzado con el usuario condor.

Si no existe usuario y grupo condor en el sistema, el administrador puede especificar que uid y gid deberían usar los demonios de Condor cuando no necesitan privilegios de root. Esto se puede hacer mediante la variable de entorno CONDOR_IDS. En cualquier caso el valor que se le asigne es el valor entero del UID, seguido del valor entero del GID.

En el equipo desde el cual se envía el trabajo, el demonio *Schedd* cambia su UID efectivo a root de tal forma que pueda inicializar al demonio *Shadow* para el trabajo que va a ejecutarse.

Antes de que el demonio *Shadow* sea creado, el demonio Schedd cambia a root con el fin de que el demonio *Shadow* sea inicializado con el UID real del usuario que emitió el trabajo.

Cabe mencionar que para asegurarse de que un trabajo no acceda a los recursos locales o cause algún daño, el mismo puede ejecutarse bajo el usuario nobody.

Dado que el demonio *Shadow* se ejecuta como el propietario del trabajo, todas las llamadas al sistema son realizadas con el UID y el GID del propietario. Esto asegura que mientras el trabajo se ejecuta acceda únicamente a los archivos a los que podría acceder el usuario si el trabajo se ejecutara localmente, sin Condor.

La variable SOFT_UID_DOMAIN es utilizada para que el middleware verifique que el UID de un determinado usuario existe en el archivo de contraseñas. Sin embargo en algunas instalaciones donde no está almacenada la contraseña de cada usuario en este archivo, el intento de ejecución del trabajo bajo Condor fallará debido a que la verificación de la contraseña ha fallado. Para evitar este problema, se configura esta variable en True para que Condor pueda ejecutar el trabajo con el UID del usuario.

3.4 Ejecución de trabajos en Condor sin privilegios de super-usuario

No siempre es posible que los demonios sean ejecutados por el super-usuario, por ejemplo cuando un único recurso configurado con Condor es compartido por varios usuarios. En este caso, por cuestiones de seguridad, puede efectuarse una instalación

donde sólo puedan emitirse trabajos por un único usuario, no habiendo necesidad de que los demonios se ejecuten por el super-usuario.

A continuación se detallan aspectos de Condor considerando los casos en los que los demonios se ejecutan por el super-usuario y donde no se ejecutan por el super-usuario.

Condor_startd: Si se está configurando un nodo para que ejecute trabajos en Condor, este demonio debe ser iniciado por el usuario root, de lo contrario los usuarios podrán iniciar, suspender, desalojar y/o eliminar los trabajos de Condor a su voluntad. De otra manera, las políticas las define el super-usuario. Si este demonio se ejecuta por el super-usuario, los trabajos se ejecutarán con un UID diferente, que puede ser el del usuario que emitió el trabajo o bajo el usuario nobody, dependiendo del valor de la variable UID_DOMAIN.

Como consecuencia de lo anterior, el trabajo que se está ejecutando no podrá dañar a los demonios de Condor.

Si los demonios no son inicializados por el super-usuario, todos los procesos iniciados por Condor, incluido el trabajo del usuario, se ejecutarán con el mismo UID, ya que el UID sólo puede ser modificado por el super-usuario.

Esto implica importantes riesgos, ya que el trabajo del usuario podría eliminar al demonio *condor_startd* y al demonio *condor_starter* ni bien se inicializa, y de esta manera evitaría ser suspendido o desalojado del recurso donde esta ejecutándose al retornar su propietario. Debido a ello, el demonio *condor_startd* siempre debe ser inicializado por el super-usuario.

Cabe destacar que si no se es super-usuario, no podrá accederse a cierta información del sistema, por lo que en estos casos el demonio *condor_startd* debe llamar a otros programas para obtener esa información. Esto resulta ser mucho menos eficiente que acceder a esta información directamente desde el kernel, que es lo que se hace cuando se ejecutan los procesos como super-usuario.

Si por alguna razón el super-usuario no puede ejecutar todos los demonios, al menos debería considerarse instalar el demonio condor_startd con el UID de root.

Condor_schedd: El mayor problema al ejecutar el demonio *condor_schedd* sin privilegios de super-usuario es que el demonio *condor_shadow* que es creado por el demonio *condor_schedd*, tendrá el mismo UID de este último lo cual significa que los usuarios que emitan sus trabajos deberán permitir acceso de escritura y lectura a los archivos y directorios, para el usuario o grupo condor, o cualquiera sea el usuario bajo el que este ejecutándose el demonio *condor_schedd*.

Condor_master: Este demonio inicia al demonio *condor_startd* y al demonio *condor_schedd*. Para que ambos se ejecuten con privilegios de super-usuario, el demonio *condor_master* debe ser iniciado por el super-usuario.

Condor_negotiator y Condor_collector: No hay necesidad de que estos demonios sean ejecutados por el super-usuario.

Condor_kbdd: La importancia de este demonio radica en que a través de él, el demonio *condor_startd* monitorea la actividad del teclado y del mouse.

Si se decide ejecutar los demonios de Condor sin privilegios de super-usuario, puede elegirse cualquier usuario, aunque lo habitual es elegir al usuario condor. Esto simplifica la configuración de Condor, ya que este usuario, por defecto, buscará los archivos de configuración en el directorio del usuario de condor.

En caso de no seleccionarse este usuario, el administrador del sistema deberá asegurarse de que Condor encuentre sus archivos de configuración necesarios.

Si los trabajos son emitidos con un usuario diferente del que está ejecutando los demonios de Condor, entonces deberá tenerse la precaución de que estos usuarios sólo puedan acceder a los archivos de su propiedad.

En la práctica suele suceder que los directorios donde se almacena la salida de los trabajos de Condor tienen permisos de escritura para todos los usuarios. Esto significa un riesgo importante en la seguridad, en el sentido de que cualquier usuario que utilice el recurso desde el cual se emitió el trabajo, podrá alterar los datos o eliminarlos. Una configuración de estas características es aceptable solo en ambientes donde los usuarios confían entre sí.

Normalmente, los usuarios sin permisos de super-usuario que necesitan utilizar Condor en sus máquinas, crean con su cuenta de usuario, un directorio home llamado condor e inician los demonios con el usuario propio.

Como en el caso donde los demonios se ejecutan con el usuario condor, no hay posibilidad de que estos cambien su UID o GID. Los demonios se ejecutan con el UID y el GID del usuario que los inició.

En un nodo desde el cual un usuario emitió un trabajo, el demonio *condor_shadow* se ejecutará bajo ese mismo usuario, pero si hay otros usuarios utilizando Condor en ese recurso, los demonios *condor_shadow* de esos otros usuarios se ejecutarán con el UID del usuario que haya iniciado los demonios.

Esto también implica un riesgo de seguridad, ya que los trabajos Condor de los otros usuarios tendrán acceso a todos los archivos y directorios del usuario que inició esos demonios.

En las instalaciones donde no existe confianza entre los usuarios, es aconsejable configurar una cuenta cuyo propietario y grupo sea condor, o permitir que cada usuario configure su propia instalación personal de Condor para emisión de trabajos.

Cuando un nodo es el sitio de ejecución para los trabajos Condor, este trabajo se ejecuta con el UID del usuario que inició al demonio *condor_startd*. Esto también implica un riesgo de seguridad, por lo que no se recomienda iniciar los demonios en el sitio de ejecución de los trabajos como un usuario regular. Se recomienda iniciar los demonios con privilegios de superusuario o bajo el usuario condor, que sólo existe para ejecutar trabajos Condor.

3.5 Ejecución de los trabajos en Condor bajo el usuario nobody

Como se ha mencionado anteriormente, los trabajos de Condor se ejecutan ya sea bajo el usuario que los emitió o bajo el usuario nobody.

Condor utiliza el usuario nobody si el valor de la variable de configuración UID_DOMAIN del nodo emisor del trabajo y el de ejecución es distinto.

Cuando Condor termina de ejecutar un trabajo, elimina todos los procesos iniciados por el trabajo, pero es posible engañar a Condor y dejar procesos activos una vez que haya realizado la limpieza. Si el trabajo se ejecuta bajo el usuario nobody, es posible dejar procesos maliciosos que esperan al próximo trabajo que se ejecute bajo el mismo usuario, pudiendo esto causar problemas.

Condor evitaría este tipo de problemas eliminando todos los procesos que se ejecuten bajo el usuario nobody, pero esto no sería oportuno, ya que los procesos del sistema que no son de Condor se ejecutan bajo el usuario nobody.

Condor ofrece una solución que consta de dos partes para resolver este problema. Por un lado, se debe crear una cuenta específica para los usuarios para que Condor las pueda utilizar, en lugar de recurrir al usuario nobody. Estas pueden ser cuentas con privilegios restringidos.

Otra solución que acompaña a las anteriores, es configurar a Condor para que esas cuentas sean utilizadas sólo por Condor, a fin de que pueda eliminar los procesos que pertenecen a ese usuario, una vez que termina el trabajo.

3.6 Directorios para los trabajos en ejecución.

Cada proceso que se ejecuta en el sistema tiene noción de su directorio de trabajo. Este directorio es la base para el acceso a todo el sistema de archivos. Existen dos directorios de trabajo para todo trabajo en Condor: uno desde el cual se emite el trabajo y otro donde se ejecuta el trabajo.

Cuando un usuario emite un trabajo, el directorio desde el cual se emite es el mismo que utiliza el usuario cuando ejecuta el comando *condor_submit*. Este directorio permanece sin cambios durante toda la vida del trabajo. El directorio de trabajo del lado emisor del trabajo es también el directorio del demonio *condor_shadow.*

Esto es particularmente importante para los trabajos que se ejecutan en el universo Standard, ya que el acceso al sistema de archivos para ese trabajo se realiza a través del demonio *condor_shadow*.

Existe también un directorio del lado de ejecución de los trabajos, que para el universo Standard es el subdirectorio execute del directorio home de Condor. Este directorio tiene permisos de escritura para todos los usuarios.

Normalmente, los trabajos que se ejecutan en el universo standard no accederán a este directorio, ya que todas las llamadas al sistema para Entrada/Salida son enviadas directamente al demonio *condor_shadow* en la máquina emisora. En el caso de que un

trabajo se corrompa y se cree un archivo con el error, el directorio de trabajo del lado de emisión debe ser accesible para el trabajo, a fin de que pueda escribir en este archivo.

El archivo con el reporte del error se envía a la máquina desde la que se emitió el trabajo y el demonio *condor_shadow* es informado de este evento, el cual envía un e-mail al propietario del trabajo notificando de la existencia de este archivo.

3.7 Configuración básica de seguridad

Condor brinda soporte para autenticación, encriptación e integridad, así como también implementa mecanismos de autorización. Sin embargo, Condor no habilita estos mecanismos por defecto por dos razones. Primero, implementar mecanismos fuertes de autenticación y autorización requiere de soporte extra fuera del ambiente de Condor, tales como los certificados GSI, o un Centro de distribución de Claves Kerberos (KDC).

Segundo, las versiones anteriores de Condor no soportaban mecanismos fuertes de seguridad, por lo que la configuración por defecto de Condor funciona correctamente cuando interactúan versiones antiguas con versiones nuevas.

La instalación por defecto de Condor está basada en seguridad a nivel de host. Esta forma de seguridad permite el acceso basándose en direcciones IP. Por ejemplo Condor se puede configurar para permitir la emisión de trabajos de cualquier usuario del dominio ejemplo.com, pero los comandos de administrador sólo pueden ejecutarse desde la máquina comando.ejemplo.com.

Debido a que la seguridad basada en host no distingue entre usuarios, cualquiera puede acceder al recurso comando.ejemplo.com y ejecutar comandos de administrador.

Es importante mencionar que Condor es susceptible a ataques donde pueda robarse la dirección IP de un recurso. Aunque la seguridad a nivel de host es limitada, es aceptable para instalaciones de poca envergadura, particularmente si los nodos están en redes privadas.

3.8 Descripción de los niveles de acceso en Condor

Es importante conocer los diferentes niveles de acceso que ofrece Condor ya que de ello depende si se podrá tener acceso a la información del pool, si se podrán ejecutar trabajos, acceder a la cola de trabajos y a los nodos del pool, entre otras actividades.

Los diferentes niveles de acceso son [22]:

Acceso READ: Los nodos con acceso Read, pueden leer información de configuración de Condor, es decir pueden ver el estado del pool, la cola de trabajos y ver los permisos de los usuarios. Un nodo con acceso Read no puede ejecutar trabajos, sólo puede visualizar información sobre el estado del pool.
Acceso WRITE: Los nodos con acceso Write pueden escribir información de Condor, ejecutar trabajos y anunciarse en el pool.

Acceso ADMINISTRATOR: Los nodos con acceso Administrator tienen derechos especiales de administrador en el pool de Condor, por ejemplo pueden cambiar las prioridades de los usuarios y reiniciar y reconfigurar archivos y demonios. El acceso administrador implica poseer acceso de lectura y de escritura.

Acceso OWNER: Este nivel de acceso se requiere para que el usuario y/o el propietario del nodo pueda ejecutar comandos de Condor, por ejemplo, el comando *condor_vacate* provoca que el demonio *condor_startd* deje vacante un trabajo Condor que esta ejecutándose en el nodo.

Acceso NEGOTIATOR: El demonio *condor_negotiator* se ejecuta en el administrador central del pool de Condor. Los comandos que necesitan de este nivel de acceso son aquellos que le solicitan al demonio *condor_schedd* que comience la negociación y los que notifican al demonio *condor_startd* que se ha producido una asociación para comenzar a ejecutar trabajos.

Acceso CONFIG: Este nivel de acceso se requiere para modificar la configuración de los demonios mediante el comando *condor_config_val*. Las máquinas con este nivel de acceso serán capaces de cambiar los parámetros de configuración, excepto aquellos parámetros especificados en el archivo de configuración *condor_config.root*.

Por defecto, el acceso CONFIG, esta denegado para todos los nodos.

Acceso DAEMON: Este nivel de acceso es utilizado por comandos internos de Condor, por ejemplo cuando el demonio *condor_startd* envía el ClassAd al demonio *condor_collector*. Este nivel de acceso sólo debe darse a los usuarios bajo los cuales se ejecutan los demonios de Condor. El acceso daemon implica acceso de lectura y escritura.

Acceso CLIENT: Este nivel de acceso es diferente de todos los demás. Mientras que lo otros niveles de acceso se refieren básicamente a políticas de seguridad para aceptar conexiones desde otros nodos, el nivel de acceso client se aplica cuando los demonios de Condor se comunican hacia otros nodos. En otras palabras, este nivel de acceso especifica la política del cliente que inicia la operación, más que el servidor al cual se está conectando.

A la hora de configurar los parámetros de seguridad en los nodos utilizados en esta tesis, se han tenido en cuenta las acciones que tienen permitidas y denegadas cada uno de ellos. Para ello se configuran dos listados, uno de permitidos (allowed) y otro de denegados (denied). Estos listados contienen los nombres o direcciones IP de los correspondientes nodos.

A su vez, se han configurado las variables HOSTALLOW_WRITE, HOSTDENY_READ, HOSTALLOW_ADMINISTRATOR con los nombres de los hosts correspondientes, teniendo en cuenta que estos listados afectan el acceso de los nodos a todos los demonios.

3.9 Negociación de los aspectos de seguridad

Dado el amplio rango de entornos y demandas de seguridad particulares, Condor debe ser flexible, lo cual se logra configurando ciertos parámetros en los archivos de configuración.

El proceso mediante el cual Condor determina los parámetros de seguridad que se utilizarán cuando se establezca una conexión se denomina negociación de la seguridad.

El principal objetivo de este proceso es determinar qué aspectos de la autenticación, la encriptación y de la integridad se habilitarán para esa conexión.

Además Condor soporta múltiples tecnologías para autenticación y encriptación, y durante el proceso de negociación de la seguridad se determina qué tecnología se utilizará para la conexión.

3.10 Configuración

El proceso de negociación de la seguridad resuelve varias combinaciones demonios-clientes en función de las características de seguridad específicas del pool. En el caso de que se quiera evitar la autenticación siempre y cuando sea posible, ésta puede configurarse como opcional, de lo contrario puede configurarse como requerida. Cabe destacar que algunas operaciones en Condor, como el envío de un trabajo siempre requerirán autenticación, más allá de la política que se implemente. Otros comandos, como *condor_q* no siempre requieren autenticación.

La configuración de la autenticación, encriptación e integridad depende de donde se desarrolle el proceso de negociación de la seguridad. En la tabla 1 se muestran los aspectos de seguridad en distintas combinaciones clientes-demonios.

Si la celda dice **Sí** significa que esta característica se puede utilizar, **No** significa que no puede utilizarse y **Falla** significa que es incompatible para esa combinación de cliente-demonio.

		Configuración del demonio			
		Nunca	Opcional	Preferida	Requerida
Cliente Configuración	Nunca	No	No	No	Falla
	Opcional	No	No	Si	Si
	Preferida	No	Si	Si	Si
	Requerida	Falla	Si	Si	Si

Tabla 1. Resolución de características de seguridad

Respecto a la habilitación de encriptación y/o al chequeo de la integridad, depende del lugar donde se realice el proceso de autenticación. La autenticación brinda el

intercambio de claves, las cuales son necesarias tanto para la encriptación como para el chequeo de integridad.

3.11 Autenticación

La autenticación es el proceso de verificar que una entidad es quien dice ser. Existen varias formas de autenticación, tales como utilizar una combinación de nombre de usuario y una contraseña o el uso de criptografía de clave pública.

En esta tesis, la autenticación de los usuarios en la máquina que emite los trabajos, se basa en políticas estándares de cuentas de sistemas operativos.

Cuando un usuario quiere acceder al pool para procesar trabajos, Condor utiliza un identificador de usuario y de grupo (user ID, group ID) para determinar quién es ese usuario. En el caso particular del usuario condor, el identificador de usuario y de grupo es configurado a 2.2

Una vez que el usuario es autenticado, Condor necesita conocer los privilegios que tiene ese usuario. Para ello, existe un archivo de configuración general de Condor llamado condor_config, en el cual se fija el valor de la variable HOSTALLOW_READ=*, indicando que todos pueden ver el estado del pool, pero no pueden unirse o ejecutar trabajos en el mismo.

Luego se configura la variable HOSTALLOW_WRITE para que únicamente las máquinas dentro del dominio de interés puedan emitir trabajos y unirse al pool.

Condor soporta varios protocolos de autenticación, como Kerberos [23], GSI [24] y Secure Socket Layer (SSL) [25]. También hace uso de otros mecanismos de autenticación, como contraseñas, autenticación de sistema de archivos, autenticación remota de sistema de archivos, autenticación para Windows, autenticación “Climed to be” y autenticación anónima [26].

Grid Security Infrastructure (GSI)

Grid Security Infrastructure (GSI) es un protocolo de seguridad basado en infraestructura de clave pública. Fue desarrollado como parte del Toolkit de Globus [19]. En este apartado se explica el procedimiento de autenticación utilizando GSI.

Un usuario A se autentica con otro usuario B, en este caso A es el cliente y B es el demonio dentro de la comunicación. En este caso, B verifica la identidad de A mediante el certificado que A le otorga, haciendo uso de la Autoridad Cerficante (CA) de B. El cliente A otorga un certificado al demonio B.

A su vez, B verifica la validez del certificado y verifica que la Autoridad Certificante que firmó el certificado de A es una en la que él puede confiar.

El protocolo de autenticación GSI necesita de un certificado X.509. El formato estándar de los certificados X.509 está dado por la Internet Engineering Task Force [27], donde

cada usuario o servicio que necesita ser identificado, tendrá un certificado X.509 en el cual se indica su nombre, su clave pública, un período de validez, una identidad y firma de la entidad emisora, una Autoridad Certificante y la clave privada correspondiente a la clave pública del certificado.

Para que Condor pueda implementar este tipo de autenticación, necesita conocer la ubicación del certificado del cliente A y del listado de Autoridades Certificantes de B.

Cuando un lado de la comunicación, ya sea el cliente A o el demonio B es un demonio de Condor, estas ubicaciones se determinan mediante configuración o se establecen por defecto.

Secure Socket Layer (SSL)

Este método de autenticación es similar a GSI pero sin la posibilidad de delegación de Proxy. SSL también utiliza certificados X.509.

El procedimiento de autenticación en Condor mediante SSL es mutuo, es decir que cuando un proceso se comunica con otro y se está utilizando SSL, cada proceso debe ser capaz de verificar la firma del certificado ofrecido por el otro proceso.

El proceso que inicia la conexión se denomina cliente y el proceso que recibe la conexión es el servidor. Por ejemplo cuando el demonio *condor_startd* se autentica con el demonio *condor_collector* a fin de entregarle el ClassAd de la máquina, el demonio *condor_startd* inicia la conexión y actúa como el cliente, mientras que el demonio *condor_collector* actúa como el servidor.

Los nombres y ubicaciones de las claves y certificados para los clientes, servidores y archivos utilizados para especificar las Autoridades Certificantes, se definen mediante archivos de configuración.

Kerberos

Kerberos es un protocolo de autenticación que hace uso de un tercero de confianza llamado servidor Kerberos que permite identificar la identidad de un usuario o un servicio.

El servidor Kerberos lleva un registro de todas las entidades con sus claves privadas, que distribuye a los dueños, en el momento de la registración. Kerberos brinda autenticación tanto para las entidades y los usuarios de los servicios como para los mensajes que estos envían en la red.

En Condor, la autenticación mediante Kerberos se habilita incluyéndolo en la lista de métodos de autenticación en el archivo de configuración, y especificando el archivo que asocia los nombres en la base de datos del servidor Kerberos con las identificaciones de los usuarios utilizados en el dominio en el cual opera Condor. También debe especificarse la ubicación de las claves privadas.

Encriptación

La encriptación brinda soporte para la privacidad entre dos partes que van a comunicarse. En Condor, tanto el usuario de los recursos como los demonios del middleware pueden especificar, mediante la configuración específica de macros, si la encriptación es obligatoria para las comunicaciones futuras.

3.12 Autorización

Los mecanismos de autorización protegen a los recursos permitiendo o denegando las solicitudes de acceso de los usuarios a esos recursos. La autorización puede definirse en términos de usuarios o basada en hosts, siendo más recomendable implementar mecanismos de autenticación basados en nombres de usuarios completamente calificados.

El mecanismo de autenticación de Condor se basa en la configuración de macros. Estas macros indican el nivel de acceso que tiene cada usuario y cada nivel de acceso tiene su propia lista de usuarios.

Durante una comunicación cada cliente puede especificar su propia lista de servidores de confianza. Cabe destacar que en los métodos de autenticación Claimed To Be y de Sistema de Archivos no son métodos simétricos. En estos casos, el cliente es autenticado por el servidor y no en el otro sentido.

Las variables ALLOW_CLIENT y DENY_CLIENT indican para cada cliente la lista de servidores en los que confía y en los que no. Este listado es un conjunto de nombres completamente calificados, separados por coma. Por ejemplo cada usuario completamente calificado se describe de la siguiente manera:

username@domain/hostname

Donde la información que se brinda del lado izquierdo de la coma representa a un usuario dentro de un dominio. Del lado de la barra se indica el nombre del host desde el cual el usuario podrá ejecutar comandos de Condor. Este nombre puede ser una dirección IP o un nombre completamente calificado.

3.13 Sesiones seguras en Condor

Con el fin de establecer comunicaciones seguras en Condor pueden implementarse mecanismos de autenticación, encriptación e integridad. La implementación de mecanismos de autenticación fuertes consume una importante cantidad de tiempo. Además, la generación de claves criptográficas para encriptación y controles de integridad pueden consumir bastante potencia de cálculo.

Condor utiliza el concepto de sesiones con el fin de almacenar información de seguridad relevante, como por ejemplo las claves de los demonios que se han autenticado previamente. De esta manera se puede agilizar el establecimiento de comunicaciones seguras entre los demonios de Condor Se establece una nueva sesión al crearse una conexión por primera vez entre un demonio y otro.

Cada sesión tiene un tiempo de vida fijo después del cual expira y se necesitará establecer una nueva sesión, pero mientras exista podrá utilizarse tantas veces como se necesite. Cada entidad de esa sesión tendrá acceso a una clave de sesión que prueba la identidad de la entidad que se encuentra en el otro extremo de la conexión.

Para poder intercambiar esta clave, se necesita implementar un mecanismo de autenticación fuerte como Kerberos o GSI, los otros métodos de comunicación mencionados no soportan intercambio de claves.

Es recomendable implementar los mecanismos de integridad y encriptación si se ha implementado el mecanismo de autenticación, de lo contrario, es posible que un atacante utilice una sesión abierta para conectarse con un demonio con fines maliciosos.

Para determinados demonios, el tiempo de sesión queda establecido en la instalación por defecto de Condor, pero también puede configurarse este tiempo mediante macros del archivo de configuración, lo cual no es lo recomendable.

3.14 Configuración de la seguridad en Condor basada en host

La implementación de seguridad basándose en host es una forma anticuada, sólo se mantiene documentada por cuestiones de compatibilidad con versiones anteriores y puede combinarse con la autorización basada en usuario.

3.15 Condor y firewalls

Como ya se ha explicado, los nodos en Condor cumplen distintas funciones, y uno de los más importantes es el nodo administrador central. Un grupo de nodos que comparten el mismo administrador central constituye un pool. Los pools pueden unirse mediante el mecanismo de flocking, como se explicó en la subsección 2.4.1. En este caso, el administrador central actúa como un broker de recursos para todo el pool, controlando la ejecución de los trabajos, la disponibilidad de recursos, los nodos que están ejecutando trabajos, etc.

Esto significa que todos los nodos del pool deben poder comunicarse con el administrador central y vice-versa. Sin embargo, hay algunas excepciones:

- Pueden haber nodos que sólo emitan trabajos pero nunca los ejecuten. Estos nodos no necesitan comunicarse entre ellos, pero sí con el administrador central y con los nodos que ejecutan trabajos, por lo tanto la comunicación es

bidireccional.

- Pueden haber nodos que sólo ejecuten trabajos pero que nunca los emitan, entonces estos nodos no necesitan comunicarse entre sí pero deben poder comunicarse con el administrador central y con los nodos que emiten trabajos. En este caso la comunicación también es bidireccional.

En la figura 9 se observa la arquitectura de pool:

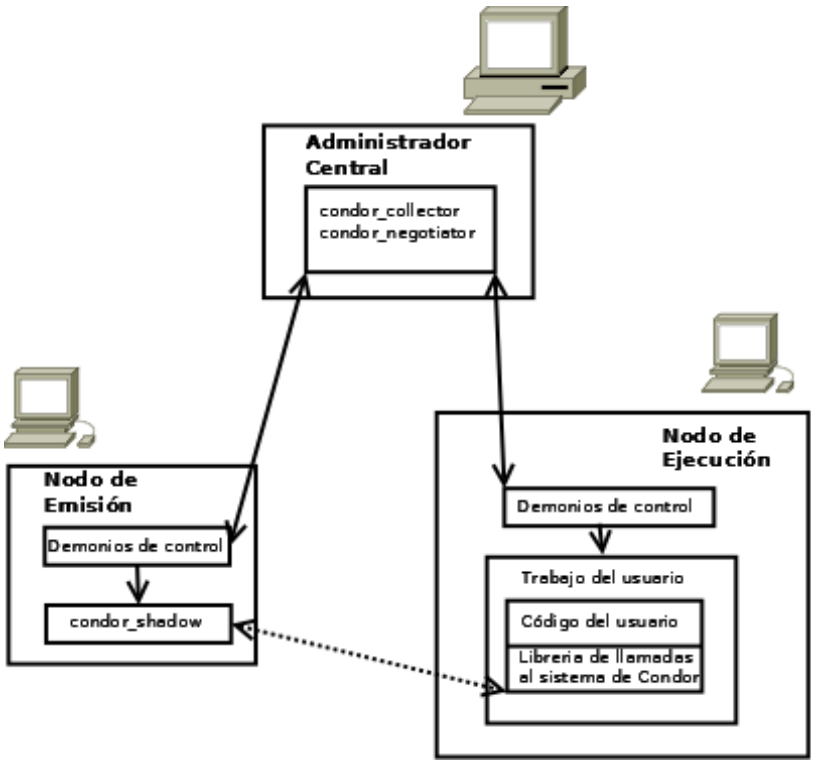

Figura 9. Arquitectura de un pool de Condor

En el caso de que los pools estén vinculados mediante el mecanismo de flocking, el escenario sería el siguiente:

- Los nodos que deban enviar un trabajo fuera de su propio pool necesitan comunicarse con el administrador central del pool en el cual ejecutará el trabajo.
- Cualquier nodo que emita un trabajo necesita comunicarse con el nodo que ejecutará ese trabajo, por lo que la comunicación es bidirieccional. Esto quiere decir, en definitiva, que todas las máquinas de cada pool deben poder comunicarse con todas las máquinas de los demás pools.
- Los nodos que sólo emiten trabajos necesitan comunicarse con el administrador central del pool donde se ejecutarán los trabajos y con los nodos que ejecutarán esos trabajos.
- Los nodos que ejecutan trabajos deben tener comunicación bidireccional con

cualquier máquina que le envíe trabajos para ejecutar, sin importar el pool al que pertenezcan.

- El caso más general es cuando cada nodo necesita comunicarse con cualquier otro nodo, ya sea que estén en el mismo pool o mediante el mecanismo de flocking.

Es importante destacar que la mayoría de las comunicaciones en Condor se producen sobre UDP más que con TCP, sobre todo a la hora de atravesar un firewall, ya que las reglas suelen ser distintas para un protocolo u otro.

Condor utiliza el puerto 9618 para el demonio *condor_collector* en el administrador central, pero puede seleccionarse otro número de puerto en el archivo de configuración global.

Para que Condor pueda atravesar un firewall, es necesario que estén abiertos algunos puertos para todos los nodos que sean parte del pool:

- El puerto 9618 para tráfico bidireccional, tanto para TCP como para UDP.
- Un rango de puertos lo suficientemente amplio y por encima del puerto 1024, para tráfico UDP y TCP. La configuración por defecto de Condor ofrece el rango de puertos desde el 9600 al 9700.

Hay tres mecanismos para utilizar firewalls con Condor:

- Restringir a Condor para que utilice un rango determinado de puertos y permitir que las conexiones a través del firewall utilicen cualquier puerto dentro de ese rango.
- Utilizar los mecanismos Condor Connection Brokering (CCB) o Generic Connection Brokering (GCB).
- Configurar una Red Privada Virtual (VPN) para atravesar el firewall. Esta alternativa se ha implementado como un aporte de esta tesis y se desarrolla en la sección 3.16

En el caso de que un pool de Condor esté totalmente detrás de un firewall, no es necesario definir un rango de puertos. Sin embargo, en el caso que el firewall se encuentre dentro de un pool de Condor, entonces habrá que configurar algunas variables para forzar el uso de determinados puertos y rango de puertos.

Por defecto, Condor utiliza el puerto 9618 para que el demonio *condor_collector* y puertos elegidos aleatoriamente para los demás demonios.

Para definir el rango de puertos que utilizará Condor, deben configurarse las variables HIGHPORT y LOWPORT.

Condor Connection Brokering (CCB) [28] permite la comunicación entre componentes donde uno de ellos se encuentra dentro de una red privada o detrás de un firewall. Por ejemplo, un proceso A de Condor necesita comunicarse con otro proceso B, pero la red no permite que se creen conexiones TCP desde A hacia B sino desde B hacia A. En este caso se puede configurar a B para que se registre en un servidor CCB al cual pueden

conectarse A y B. Entonces, cuando A necesita conectarse con B, le envía una solicitud al servidor CCB, el cual a su vez le indicará a B que se conecte con A.

Condor Connection Brokering no soporta la ejecución de trabajos del universo estándar.

Actualmente se está reemplazando GCB [29] con CCB. Básicamente, GCB permite la comunicación entre nodos que están dentro de redes privadas. Se prefiere CCB a GCB porque soporta todas las plataformas, incluyendo Windows, es más sencillo de configurar y para resolver problemas y puede reiniciarse y reconfigurar sobre la marcha, mientras está en uso.

GCB posee varias ventajas pero su configuración es muy complicada. Otras desventajas importantes son:

- El recurso que actúe como GCB representa un punto central de falla.
- Todos los nodos que se encuentran detrás del firewall comparten una única dirección IP, que es la IP pública de su broker GCB. Esto es importante desde el punto de vista de la seguridad y representa cierta complejidad a la hora de realizar un debug del sistema.
- GCB no puede implementar el mecanismo de seguridad Kerberos.
- Es complicado establecer las conexiones.
- Cada puerto de cada nodo que se encuentra dentro de la red privada debe corresponder a un único puerto en el nodo que actúe como broker, por lo que hay un número limitado de nodos privados a los que el broker puede darle servicio.
- Cada nodo debe mantener una conexión TCP con su broker. En caso de que el socket se cierre, el broker intentará mantenerlo abierto lo que significa que deberá tener tantos sockets abiertos como nodos privados a los que les da servicio.

3.16 Configuración de Condor para Redes Privadas Virtuales (VPNs)

Una forma de resolver la conectividad de Condor a través de un firewall es mediante la implementación de Redes Privadas Virtuales (VPNs) entre las diferentes partes de un pool de Condor o entre diferentes pools.

Como aporte de esta tesis, se comprueba el comportamiento de Condor a través de una VPN, para lo cual se diseñó una maqueta para interconectar dos pools de Condor configurados en dos redes Lan vinculadas mediante dos routers, como se muestra en la figura 6. Otro aporte de esta tesis resulta en el estudio del overhead que se agrega a los paquetes de Condor cuando atraviesan la VPN.

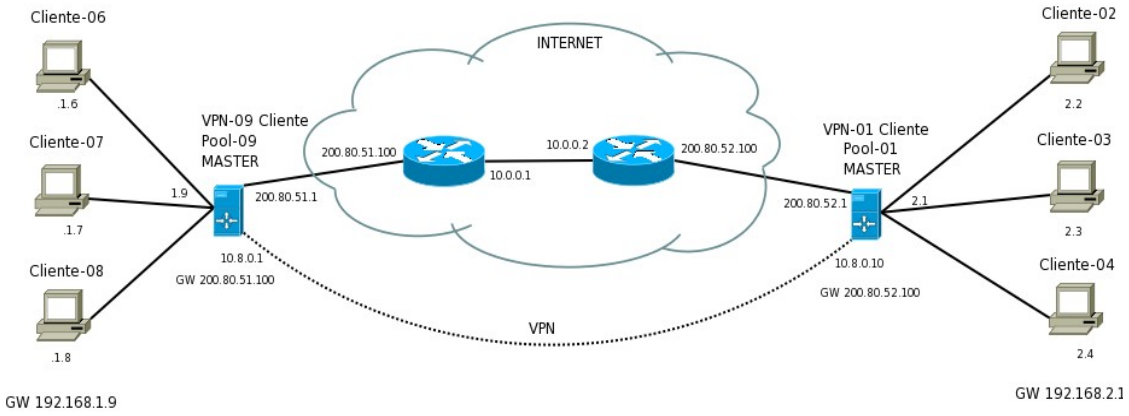

Figura 10. Conexión de dos pool de Condor mediante una VPN

Las redes privadas virtuales son conexiones seguras entre redes privadas, materializadas sobre una infraestructura de acceso público. Las VPNs implementan típicamente combinaciones de encriptación, certificados digitales, autenticación fuerte de usuario y control de acceso. Las VPNs conectan numerosos nodos detrás de un firewall o de un gateway.

La red privada virtual de la figura 10 se implementó con OpenVPN [30], el cual es un demonio VPN robusto y altamente configurable, que puede utilizarse para segurizar enlaces entre dos o más redes, mediante un túnel encriptado sobre Internet. OpenVPN utiliza SSL como mecanismo de seguridad subyacente. Entre sus características se puede mencionar:

- Puede tunelizar cualquier subred IP sobre un puerto UDP o TCP y crear túneles entre cualquier sistema operativo que soporte OpenVPN (Windows 20000/XP, Linux, Solaris, FreeBSD, OpenBSD, NetBSD, y MacOS X).
- Tiene licencia open source (GPL).
- Utiliza los mecanismos de encriptación, autenticación y certificación que provee la biblioteca OpenSSL para proteger el tráfico de red que atraviesa Internet.
- Puede configurarse para utilizar encriptación de llave estática o encriptación de llave pública basada en certificado.
- Implementa mecanismos para administrar la utilización del ancho de banda.
- Tuneliza redes sobre NAT.
- Es sencillo de configurar y probar.

OpenVPN se diseñó principalmente para operar sobre redes no seguras y para responder oportunamente ya sea durante la operación normal del sistema como durante la recuperación de errores.

En la figura 6 se observa que a las dos redes locales se les ha asignado el rango de direcciones IP 192.168.218.1.X y 192.168.218.2.X y OpenVPN asigna una dirección IP privada de este rango a cada uno de los puntos finales de la VPN. En ambos extremos del túnel, OpenVPN recoge los datos que se han enviado a las direcciones virtuales, encapsula y encripta estos datos con las claves almacenadas localmente y luego transmite los datos a las direcciones reales de los nodos en el otro extremo de la conexión. El extremo receptor desencapsula los datos y verifica su origen y solo los datos segurizados con la llave correcta serán desencapsulados y enviados al nodo destinatario, ya que el resto será rechazado.

Finalmente el extremo receptor tratará a los datos como si hubieran sido recibidos por su dirección IP virtual, creando así un túnel seguro entre dos extremos de la VPN.

La implementación de redes privadas virtuales resulta ser una buena solución para atravesar firewalls ya que sólo un nodo que pertenece a la red protegida por el firewall necesita tener acceso externo (gateway VPN). Todos los nodos que pertenecen a la red protegida tunelizan el tráfico de Condor a través del nodo que actúa como gateway VPN y el tráfico entre distintos gateways estará encriptado, añadiendo así un mayor nivel de seguridad.

OpenVPN se puede configurar para que rutee entre redes privadas y los clientes de un gateway VPN que se encuentran en una LAN, pueden estar en la misma subred que los que se encuentran en la red LAN del otro lado de la VPN. Esta configuración resulta transparente desde el punto de vista de Internet y de Condor. Es importante destacar que Condor no participa de la configuración de la VPN y la VPN le resulta totalmente transparente.

3.16.1 Configuración de OpenVPN

Para configurar OpenVPN se deben realizar los siguientes pasos. Cabe destacar que esta configuración se llevó a cabo sobre distribuciones de Linux CentOS:

- Instalar la librería OpenSSL.
- Instalar el paquete OpenVPN.
- Establecer las direcciones IP de los nodos de las redes locales que se vincularán mediante la VPN.
- Configurar una Autoridad Certificante (CA) y generar los certificados y las claves para el servidor OpenVPN y cada uno de los clientes. OpenVPN soporta autenticación bidireccional basada en certificados, por lo que el cliente debe autenticar el certificado del servidor y el servidor debe autenticar el certificado del cliente antes de establecer una relación de confianza mutua. Tanto el cliente como el servidor se autenticarán mutuamente verificando que el certificado fue

firmado por la Autoridad Certificante, y luego comprueban la información del encabezado del certificado autenticado, el cual debe poseer un nombre común.

- Generar el certificado y la clave de la Autoridad Cetificante maestra. En este paso se crea el certificado y la clave de la Autoridad Certificante maestra, el certificado y la clave del servidor y el certificado y la clave para cada uno de los clientes.
- Una vez que se han creado las claves y certificados deberán aparecer los archivos:

Nombre del archivo	Utilizado por	Propósito	Secreto?
ca.crt	servidor + todos los clientes	Certificado root de la CA	No
ca.key	Sólo el nodo que firma la clave	Clave root de la CA	Sí
dh{n}.pem	Sólo el servidor	Parámetros Diffie Hellman	No
server.crt	Sólo el servidor	Certificado del servidor	No
server.key	Sólo el servidor	Clave del servidor	Sí
client1.crt	Sólo el cliente 1	Certificado del cliente 1	No
client1.key	Sólo el cliente 1	Clave del cliente 1	Sí
client2.crt	Sólo el cliente 2	Certificado del cliente 2	No
client2.key	Sólo el cliente 2	Clave del cliente 2	Sí
client3.crt	Sólo el cliente 3	Certificado del cliente 3	No
client3.key	Sólo el cliente 3	Clave del cliente 3	Sí
ca.key	Sólo el nodo que firma la clave	Clave root de la CA	Sí
dh{n}.pem	Sólo el servidor	Parámetros Diffie Hellman	No

- El paso final en el proceso de generación de claves consiste en copiar todos los archivos a los nodos que los necesitan. Es importante destacar que deben ser copiados de forma segura, por un medio seguro, ya que si los certificados se ven comprometidos la vpn deja de ser segura.

Creación de los archivos de configuración en el servidor y en el cliente

Archivo de configuración del servidor

1 Definir la dirección IP donde escuchará OpenVPN en el servidor.

2 Definir el puerto por el que escuchará OpenVPN.

3 Definir el protocolo que se utilizará, TCP o UDP. En el caso del experimento computacional, se estableció UDP.

4 Crear el túnel IP ruteado.

dev tun

5. Crear el certificado SSL/TSL como usuario root de la Autoridad Certificante, (CA), el certificado (cert) y de la clave privada (key). Cada cliente y el servidor deben tener su clave y su propio certificado. Todos los clientes y el servidor utilizan el mismo archivo ca.

ca /etc/openvpn/keys/ca.crt

cert /etc/openvpn/keys/server.crt

key /etc/openvpn/keys/server.key

6. Establecer los parámetros Diffie Hellman

dh /etc/openvpn/keys/dh1024.pem

7. Configurar el modo de trabajo del servidor y establecimiento de las subredes VPN. El servidor tomará la dirección 10.8.0.1 para sí mismo y el resto estará disponible para los clientes. Cada cliente podrá acceder al servidor mediante la dirección IP 10.8.0.1.

server 10.8.0.0 255.255.255.0

8. Es conveniente llevar un registro de las asociaciones de cada uno de los clientes y de la dirección IP virtual asignada. De esta manera si el servicio OpenVPN es reiniciado o deja de funcionar, los clientes podrán volverse a conectar con la misma dirección IP virtual previamente asignada.

ifconfig-pool-persist ipp.txt

9. Establecer las rutas para que los clientes alcancen las subredes privadas detrás del servidor.

push "route 192.168.1.0 255.255.255.0"

10. Asignar una dirección IP específica para cada cliente o si el cliente que desea conectarse tiene una subred privada detrás la cual debe tener acceso a la VPN, utilizar el subdirectorio “ccd” para las configuraciones de archivos específicos de cliente.

client-config-dir /etc/openvpn/ccd

route 192.168.2.0 255.255.255.0

11. Se utiliza la directiva keepalive que genera mensajes similares a un ping, los cuales son enviados en la red para que cada lado de la conexión tenga conocimiento cuando el otro extremo se ha caído. En este caso se genera un ping cada 10 segundos, y se asume que si el otro extremo no responde durante 120 segundos, se ha caído.

Se habilita la compresión en el enlace VPN. Esta configuración también debe realizarse en el archivo de configuración del cliente.

comp-lzo

12. Las opciones de persistencia intentarán evitar el acceso a ciertos recursos que puedan resultar inaccesibles luego de un reinicio, debido a la disminución de privilegios.

    persist-key

    persist-tun

13. Se crea un archivo de salida donde se refleje el estado de las conexiones actuales, las conexiones fallidas, truncadas, etc.

    status openvpn-status.log

Archivo de configuración de los clientes

1. Se especifica que se trata de un cliente y que se recibirán directivas del servidor

   client

2. Se configura el túnel IP de la misma manera que en el servidor

   dev tun

3. Se elige el protocolo mediante el cual el cliente se conecta al servidor

   proto udp

4. Se establece la dirección IP, el nombre de host y el puerto del servidor.

   remote 200.80.51.1 1194

5. Se configura al cliente para que siempre intente resolver el nombre de host del servidor OpenVPN.

   resolv-retry infinite

6. Normalmente, los clientes no necesitan vincularse a un determinado número de puerto local

   nobind

7. El cliente intenta preservar su estado después de algún reinicio

   persist-key

   persist-tun

8. Este punto en la configuración del cliente es similar a la del servidor. Cabe destacar que es conveniente utilizar un archivo .crt/.key para cada cliente. Se utiliza un único archivo ca para todos los clientes.

   ca /etc/openvpn/keys/ca.crt

   cert /etc/openvpn/keys/client.crt

   key /etc/openvpn/keys/client.key

9. Se habilita la compresión del enlace VPN. Esta configuración debe estar habilitada también en el servidor

comp-lzo

Inicio de la VPN y pruebas de conectividad inicial

Inicio del servidor de VPN

- En primer lugar hay que asegurarse que el servidor OpenVPN sea accessible desde Internet. Esto significa que hay que abrir el puerto UDP 1194 en el firewall, o el número de puerto que corresponda, o bien
- Se configura una regla para que reenvíe el puerto 1194 UDP desde el firewall/gateway hacia el nodo actúa como servidor OpenVPN
- Es importante asegurarse que la interfaz tun no esté bloqueada por el firewall.

Inicio del cliente VPN

a. Es aconsejable iniciar el servidor OpenVPN desde la línea de comandos:

   openvpn [client config file]

b. La secuencia normal de inicio del cliente de una VPN debe concluir con el mensaje

   Initialization Sequence Completed.

c. Si todo está bien debería haber conectividad desde el cliente al servidor OpenVPN

Agregado de nodos en la subred de los clientes

a. En primer lugar debe instalarse la librería OpenSSL.
b. Se instala el paquete OpenVPN.
c. Se establecen las direcciones IP de los nodos de las redes locales que se vincularán mediante la VPN.
d. Se crea la clave y el cerificado para el nuevo cliente.

   ./build-key cliente_nuevo

e. Luego se siguen los pasos indicados en el punto 7, donde se detalla la configuración del cliente.

3.16.2 Overhead en el túnel VPN

Resulta de interés analizar el overhead que se agrega a los paquetes de Condor cuando atraviesan una VPN. Los paquetes VPN suponen una sobrecarga (overhead) debido al proceso de encapsulación. Cuando se envía la carga útil sobre el túnel VPN se le

agregan encabezados y colas de varios protocolos. Además hay un overhead adicionado por el cifrador utilizado en el proceso de encriptación para segurizar el túnel. Los efectos del overhead pueden disminuirse si se utiliza algún mecanismo de compresión.

El overhead en OpenVPN depende del tipo de interfaz, protocolo de transporte, algoritmo criptográfico y mecanismo de compresión elegidos [31].

El overhead fijo que se agrega a cada paquete es de 14 bytes correspondientes al encabezado de la trama y 20 bytes del encabezado IP. El encabezado del protocolo de transporte UDP (TCP) agrega 8 (32) bytes. El algoritmo criptográfico utilizado para segurizar el túnel contribuirá a aumentar el overhead dependiendo del algoritmo. Parte del overhead incluye el hash del algoritmo de método de acceso al medio utilizado, por ejemplo MD5 (128-bits) o SHA-1 (160-bits).

OpenVPN utiliza Blowfish [32] por defecto, que es un cifrador simétrico de 128 bits. Blowfish ofrece un mejor rendimiento comparado con otros cifradores como AES, DES y 3DES y consume menor tiempo de procesamiento [33].

Respecto al protocolo elegido, TCP es un protocolo orientado a la conexión y asume que se utilizará un medio de comunicación no confiable, por lo que agrega mayor overhead que UDP.

Para el experimento computacional se implementó la interfaz TUN, para conformar un túnel IP para una VPN de capa 3. En este caso la carga útil no contiene información de encabezado de capa 2 lo que reduce el overhead en 14 bytes. Entonces el overhead total que añade OpenVPN por cada paquete transmitido, teniendo en cuenta que se configuraron interfaces TUN sobre UDP será de 69 bytes, conformados por:

- 41 bytes correspondientes a la capa de seguridad, que incluye 1 byte correspondiente a la etiqueta del paquete, 20 bytes de la firma) HMAC-SHA1, 16 bytes del vector de inicialización y 4 bytes del número de secuencia.
- 28 bytes correspondientes al overhead propio del túnel (incluye encabezado IP y UDP)

Se pudo observar que la topología de la red afecta la perfomance del sistema, así como también lo hacen los componentes de hardware como la memoria, velocidad de la CPU del router y velocidad de la red, pudiendo inclusive ocasionar cuellos de botella.

La performance del sistema también se ve afectada por el tipo de interfaz, protocolo de transporte, algoritmo criptográfico y mecanismo de compresión elegidos.

Capítulo 4

4. Administración de trabajos en Condor

En el presente capítulo se discute la administración de trabajos en Condor. Para ello, en la sección 4.1, se detalla la gestión de recursos en Condor, que convierte a una colección de estaciones de trabajo en un entorno de computación distribuida de alto disponibilidad (HTC).

En la sección 4.2 se presentan los mecanismos de planning y scheduling que ofrece Condor y la interacción entre ambos. Posteriormente, en la sección 4.3 se explica en funcionamiento del mecanismo de asociación de Condor, denominado matchmaking y las características y funciones que cumplen los ClassAds durante el proceso de asociación.

En la sección 4.4 se discute el entorno de ejecución para trabajos que ofrece Condor, donde se comenta el estado en el que se encuentra cada uno de los nodos del pool de Condor así como también las actividades que están realizando en cada momento y las transiciones posibles entre esos estados.

El envío, ejecución y monitoreo de los trabajos en Condor se analiza en la sección 4.5, junto a los posibles universos que ofrece Condor a la hora de emitir un trabajo para su ejecución. La sección 4.6 trata sobre los mecanismos para administrar las prioridades en Condor, tanto de los trabajos como de los usuarios.

En la sección 4.7 se analiza la facilidad de checkpointing que ofrece Condor el cual permite reiniciar la ejecución de un trabajo desde un determinado punto, sin necesidad de tener que ejecutarlo completo desde su inicio. El concepto de workflow, según el cual pueden ejecutarse numerosas aplicaciones interdependientes, se presenta en la sección 4.8. Particularmente, se presenta el motor de workflow de Condor denominado DAGMan que se encarga de administrar este tipo de situaciones.

4.1 Gestión de recursos en Condor

Como resultado de la disminución de la relación costo/rendimiento del hardware, es posible contar con una gran variedad de recursos adecuados para ejecutar aplicaciones en entornos de computación de alto rendimiento, HTC. Muchos de los mismos son recursos de escritorio, propiedad de usuarios interactivos y con disponibilidad parcial.

Estos recursos suelen asignarse a una persona en particular dentro de una organización para que realice sus tareas diarias. Una parte de estos recursos puede agruparse en granjas y se asignan a usuarios que hacen un uso intensivo de los mismos.

Sin embargo, a pesar de que las organizaciones han aumentado notablemente su poder computacional, sólo una pequeña parte de esta capacidad de cómputo está disponible para usuarios HTC. Para poder superar esta limitación, debería traspasarse las fronteras impuestas por los propietarios sobre sus recursos, lo cual suele no ser sencillo, ya que habitualmente éstos no quieren disminuir el rendimiento y disponibilidad de sus recursos.

Las restricciones impuestas por los propietarios de los recursos con disponibilidad parcial suelen ser complejas y dinámicas, limitando la disponibilidad de los mismos. Por lo tanto es tarea del administrador de recursos conocer el estado actual de los mismos y tratarlos como recursos con poder de cómputo que pueden utilizarse de manera oportunística. Cabe destacar que estos recursos con disponibilidad parcial pueden ser reclamados por sus propietarios en cualquier momento.

Otro aspecto a considerar es la necesidad de superar los dominios administrativos de los propietarios de los recursos. El principal obstáculo que se presenta al ejecutar trabajos entre distintos dominios administrativos es el acceso a los archivos de entrada y salida que necesita el trabajo en cuestión. Este aspecto debe ser resuelto por el entorno HTC. De esta manera, un entorno HTC como Condor, ofrece al usuario una extensión de la capacidad de procesamiento.

Normalmente los entornos como Condor siguen el paradigma maestro/esclavo, donde un recurso maestro es el encargado y responsable de supervisar el trabajo de los recursos esclavos. Estos recursos esclavos pueden estar distribuidos en distintos pools pudiendo ofrecer dedicación parcial o total a la hora de ejecutar los trabajos.

Un aspecto fundamental para los usuarios que envían trabajos para su ejecución, es contar con un conjunto robusto de recursos, los cuales no deben fallar, especialmente durante la ejecución de los trabajos. Es decir, la robustez del entorno a la hora de administrar recursos para completar exitosamente la ejecución de un trabajo es más relevante que la eficiencia del mismo.

El núcleo de un entorno HTC es el Sistema de Administración de Recursos (RMS), el cual administra los recursos distribuidos a lo largo de distintos pool. Este sistema brinda servicios de administración de recursos a su comunidad de usuarios, la cual está conformada por cuatro tipos: propietarios de los recursos, administradores del sistema, programadores de aplicaciones y clientes. Cabe destacar la importancia de los propietarios de los recursos cuando se trata de recursos con disponibilidad parcial, ya que el RMS deberá garantizarle acceso al mismo sin que perciba degradación en su disponibilidad ni rendimiento al momento de utilizarlos.

Por otra parte, los administradores del sistema confían en la robustez del RMS al momento de ejecutar trabajos sin que exista la necesidad de su intervención frecuente. Los clientes de los recursos por su parte, esperan un sistema flexible y que se adapte a sus requerimientos.

El éxito de un RMS se logra cuando el mismo se ejecuta de forma continua y confiable en modo producción, con propietarios de recursos y clientes satisfechos por la calidad de servicio entregada y, por otro lado, con los administradores del sistema y programadores de aplicaciones conformes con la robustez y confiabilidad del sistema. Estos requerimientos mencionados sugieren que los Sistemas de Administración de Recursos se constituyan como un modelo en capas, donde el monitoreo y control de los

recursos se lleva a cabo en capas inferiores y las interfaces para los clientes en capas superiores [34].

Condor, como exponente de los sistemas HTC, puede administrar grandes colecciones de recursos distribuidos. La arquitectura del middleware Condor está estructurada para brindar servicios de administración de recursos ya sea a los mismos recursos, a los clientes y a las aplicaciones, las cuales pueden ser secuenciales o paralelas.

En el caso de Condor, los recursos están representados por los Resource-owner Agents (RAs), los cuales son responsables de implementar las políticas de los propietarios de los recursos.

Los RA examinan periódicamente los recursos para determinar su estado actual. Esta información es encapsulada en un ClassAd [35] junto con la política de uso del propietario del recurso.

Los clientes de Condor están representados por Customer Agents (CAs) los cuales mantienen en colas los trabajos emitidos por cada cliente, y están representados por una lista de ClassAds.

Los RAs y los CAs envían periódicamente los ClassAds al administrador del pool de Condor, describiendo los recursos y colas de trabajo respectivamente. El ClassAd del recurso y el ClassAd de la solicitud de recurso conforman el protocolo de anunciación mediante ciertas expresiones, como se verá en la sección 4.3.

Periódicamente, el administrador del pool de Condor entra en un ciclo de negociación que involucra un algoritmo de asociación encargado de determinar cual CA requiere servicio de asociación de solicitudes con recursos [36].

La arquitectura de Condor brinda funcionalidades similares a los sistemas de encolamiento tradicionales tipo batch, incorporando un mecanismo único que le permite desenvolverse muy bien en entornos donde otros sistemas de administración de recursos son más débiles.

El objetivo de la computación de alto rendimiento HTC es brindar poder computacional con tolerancia a fallos durante largos períodos de tiempo, utilizando todos los recursos disponibles de la red. Por otro lado, el objetivo de la computación oportunística es utilizar los recursos siempre que estén disponibles, y no necesariamente deben estarlo al 100%. En Condor, ambos objetivos se acoplan naturalmente. Algunos de los mecanismos de Condor que permiten este acoplamiento son:

- **ClassAds:** El mecanismo de ClassAd brinda un marco muy flexible para materializar la asociación de las solicitudes de recursos para ejecutar trabajos con las ofertas de recursos disponibles. Los ClassAds permiten a Condor adoptar prácticamente cualquier requerimiento de utilización de recurso y adoptar un mecanismo de planificación cuando se incorporan recursos a la Grid.

- **Checkpoint y migración de trabajos:** Para cierto tipo de trabajos, Condor puede efectuar de manera transparente un punto de control (checkpoint) y posteriormente reiniciar la aplicación desde un archivo de checkpoint.

El mecanismo de checkpointing periódico permite implementar tolerancia a fallos y salvaguarda el tiempo de cómputo acumulado de un trabajo. Este mecanismo permite además migrar un trabajo desde un nodo a otro.

- **Llamadas remotas al sistema:** Cuando se ejecutan trabajos en equipos remotos, Condor preserva el entorno de ejecución local mediante las llamadas remotas al sistema. Las llamadas remotas al sistema son uno de los mecanismos de Condor para redireccionar toda las llamadas de Entrada/Salida de los trabajos de vuelta al nodo que emitió el trabajo. Este mecanismo se denomina en Condor mobile sandbox.

En definitiva, Condor convierte a una colección de estaciones de trabajo en un entorno de computación distribuida de alto disponibilidad (HTC).

Como la mayoría de los sistemas por lotes, Condor brinda mecanismos para encolamiento de trabajos, políticas de planificación, esquema de prioridades y clasificación de recursos.

Los usuarios envían su trabajo a Condor, el cual los encola para su posterior ejecución e informa los resultados al usuario.

Condor puede describirse como un administrador de recursos del sistema y un planificador distribuido de trabajos que utiliza el poder computacional de un pool de computadoras comunicadas mediante una red. En este sentido, los usuarios envían trabajos como programas ejecutables y Condor administra la ejecución de esos trabajos sobre recursos elegidos adecuadamente, basándose en los requerimientos de los trabajos, propiedades de los recursos y políticas de distribución.

Así, Condor se diferencia de los sistemas de planificación tradicionales para trabajos de cómputo intensivo al no requerir que los recursos subyacentes sean dedicados. Condor seleccionará los nodos sobre los cuales se ejecutarán los trabajos y que no están siendo utilizados en ese momento de acuerdo a las políticas dictadas por el propietario de ese nodo.

Esto asegura que los recursos infrautilizados sean adecuadamente aprovechados y puedan desalojar y migrar trabajos cuando el nodo lo requiera, protegiéndolos de acuerdo a las políticas de sus propietarios.

El rango de servicios brindados por Condor es muy variado pero se puede simplificar su funcionamiento en las siguientes tres categorías:

- **Planificación de trabajos:** El planificador de trabajos es responsable de administrar las solicitudes de los trabajos para su ejecución, lo cual implica manejar archivos de entrada, requerimientos de entorno y otros parámetros, que a su vez pueden pertenecer a múltiples usuarios, y también de mantener una cola persistente de trabajos. Los usuarios pueden asignar diferentes prioridades a sus trabajos y especificar dependencias de flujos de trabajo (workflows) entre distintos trabajos.

- **Servicios de administración de recursos:** El administrador central del pool de Condor es responsable de relevar las características de los recursos y su grado de utilización. Basándose en esta información y en las prioridades de los usuarios, el administrador central asocia las solicitudes de trabajos con los recursos adecuados para su ejecución.

- **Administración de la ejecución de los trabajos:** Condor administra la ejecución remota de trabajos mediante mecanismos de transferencias de archivos para que los archivos que se necesitan para la ejecución del trabajo estén disponibles en la máquina remota, mecanismos de checkpointing para "salvar" el estado del trabajo a fin de concluir su ejecución posteriormente, y migración de trabajos de una máquina a otra.

4.2 Planning y Scheduling

Usualmente, el número de trabajos a ejecutar es superior a la cantidad de recursos disponibles, por lo tanto hay que decidir cómo asignar los recursos a los trabajos. Este proceso en Condor se conoce como scheduling. Se ha investigado mucho sobre scheduling (lo que ha motivado la proliferación de procesadores masivamente paralelos (MPP)) y la necesidad de utilizarlos de la manera más eficiente posible.

El concepto de scheduling consiste en la administración de un recurso por parte de su propietario, al cual le interesa mejorar métricas del sistema tales como la eficiencia, utilización y rendimiento sin perjudicar al usuario del recurso al que está sirviendo.

El concepto de planning involucra la adquisición de recursos por parte de los usuarios, los cuales suelen interesarse en mejorar ciertas métricas como ser tiempos de respuesta, tiempo de round-trip y rendimiento de sus trabajos, todo esto a un costo razonable.

Condor puede implementar el mecanismo de scheduling apropiativo sobre recursos dedicados de un cluster. Para ello, cuando un nodo de cluster no ha sido elegido para ejecutar un trabajo, Condor lo puede utilizar de manera oportunística, pero si ese nodo se llega a reservar para uso futuro, Condor se apropiará del trabajo que estaba ejecutando.

En definitiva, cada estación de trabajo es propiedad de un individuo que ejerce completo control sobre el recurso, lo que finalmente resulta en una "propiedad distribuida". Las preferencias personales y el hecho que las estaciones de trabajo puedan ser actualizadas y configuradas según deseen sus propietarios, deriva en un conjunto de recursos totalmente heterogéneos.

Por otro lado, los propietarios de los recursos pueden encender y/o apagar las estaciones de trabajo cuando lo deseen, dando lugar a un pool de recursos dinámico [36].

4.3 Matchmaking y ClassAds

En el proceso de matchmaking las entidades que solicitan u ofrecen servicios, anuncian sus características y requerimientos en ClassAds o avisos clasificados. El servicio de asociación designado, es decir el Matchmaker, asocia los ClassAds de manera que se satisfagan los requerimientos especificados. Luego informa a las partes intervinientes de la asociación efectuada. La intervención del Matchmaker en lo que respecta a la asociación termina en ese momento. Posteriormente, las entidades asociadas establecen contacto, negocian algunos términos y finalmente cooperan para llevar a cabo el servicio solicitado.

El entorno para lograr la asociación entre recursos y trabajos puede descomponerse en cinco partes:

- La especificación de los ClassAds, las cuales poseen un lenguaje para expresar las características y restricciones y establece la semántica para evaluar estos atributos.
- El protocolo de notificación, el cual define la relación entre los ClassAds y el estado del sistema, en lo que se refiere al proceso de asociación.
- El protocolo de asociación, encargado de definir el método de notificación a las entidades asociadas.
- El protocolo de reclamo, el cual define las acciones que deben tomar las entidades asociadas para llevar a cabo el servicio solicitado.

El Matchmaker de Condor actúa asociando los ClassAds de los trabajos y de los nodos y se asegura que todos los requerimientos en ambos ClassAds sean satisfechos.

Los trabajos de Condor y los nodos se anuncian al Matchmaker, que es parte del nodo maestro de Condor, el cual efectúa la asociación basándose en algoritmos de negociación y notifica a los trabajos y a los nodos. Los trabajos y los nodos se solicitan entre ellos y se comunican directamente mientras el trabajo está en ejecución. Si el propietario del nodo que ha sido solicitado por un trabajo lo reclama, el trabajo (dependiendo del universo) pasa a la instancia de checkpointing, anuncia su estado al Matchmaker y espera a ser planificado en un nuevo nodo que satisfaga los requerimientos del trabajo. En la figura 11 se visualiza el proceso de matchmaking:

Figura 11. Proceso de matchmaking en Condor

Los ClassAds son el elemento central del mecanismo de matchmaking o asociación de recursos con requerimientos de los trabajos en Condor.

Estos avisos clasificados son un modelo semi-estructurado y utilizan un esquema libre, permitiendo asociar atributos y expresiones que van a ser contrastados con otro aviso clasificado.

Un aviso clasificado expresa las características de los trabajos de los usuarios, así como los requerimientos y preferencias para asociar ese trabajo a un recurso determinado que cumpla con esas características.

Los nodos en un pool de Condor anuncian sus atributos, tales como memoria disponible, tipo y velocidad de CPU, tamaño de memoria virtual, carga promedio actual, etc. Este ClassAd de nodo también informa sobre las condiciones bajo las cuales ejecutará los trabajos y qué tipo de trabajos prefiere ejecutar.

Por otro lado, cuando se emiten los trabajos se especifica un ClassAd con los requerimientos y preferencias para ese trabajo. El ClassAd incluye el tipo de nodo que se desea utilizar para ejecutar el trabajo y las características que debe ofrecer. En la figura 12 el ClassAd correspondiente a un trabajo y en la figura 13 el ClassAd correspondiente a un recurso.

```
Type = “Job"
TargetType = “Machine"
Requirements = ((other.Arch==\INTEL" && other.OpSys==\LINUX") && other.Disk > my.DiskUsage)
Rank = (Memory * 10000) + Kflops
Department = “ITU”
Cmd = “run-sim"
DiskUsage = 6000
```

Figura 12. ClassAd correspondiente a un trabajo

```
Type = “Machine"
TargetType = “Job"
Machine = “nodo1.itu.uncu.edu.ar"
Requirements = (LoadAvg <= 0.300000) && (KeyboardIdle > (15 * 60)) Rank = other.Department==self.Department
Arch = “INTEL"
OpSys = “LINUX"
Disk = 3076076
```

Figura 13. ClassAd correspondiente a un recurso

Debido a que los ClassAds son de esquema libre y a fin de evitar errores por parte de los usuarios, utilizan una lógica de tres valores, la cual puede ser evaluada en True, False y Undefined.

El Matchmaker de Condor otorga significados especiales a dos atributos especiales de los ClassAds, Requirements y Rank. El atributo Requirements indica una limitación, mientras que Rank indica una preferencia dentro de esa limitación.

Para poder asociar dos ClassAds, el algoritmo de asociación necesita que las expresiones Requirements de ambos ClassAds sean evaluadas en True. El atributo Rank es utilizado para elegir entre asociaciones compatibles, es decir entre el ClassAd del recurso y el ClassAd del requerimiento, el Matchmaker elige aquel con el valor de Rank más alto. Cabe destacar que el valor del atributo Rank es un número arbitrario en punto flotante.

En el ejemplo de la figura 12, el atributo Requirements, establece que el trabajo debe ser asociado con una máquina Intel con sistema operativo Linux y que tenga suficiente

espacio en el disco rígido (más de 6 megabytes). De todas las máquinas que cumplan con ese requerimiento, el trabajo preferirá a aquellas que tengan mucha memoria y buen rendimiento en punto flotante.

Por otro lado, en la figura 13, el atributo Requirements, establece que ese recurso no se asociará para ejecutar ningún trabajo a menos que su load avarage sea bajo y el teclado haya estado ocioso por más de 15 minutos. Una vez que el recurso decida ejecutar un trabajo, el atributo Rank establece que prefiere ejecutar trabajos de los miembros del mismo Departamento al que él pertenece [1].

Los ClassAds se consideran la lengua franca de Condor, es decir el lenguaje utilizado para describir los trabajos y las características de los recursos disponibles. Los ClassAds son intercambiados por los procesos de Condor a la hora de planificar los trabajos y se guarda información en archivos de log para realizar estadísticas y debugging.

Como se mencionó en la sección 4.1, los nodos que ejecutan trabajos en Condor son representados por Worker Agents (WAs), que son los encargados de hacer cumplir las políticas establecidas por sus propietarios y los clientes de Condor son representados por Customer Agents (CAs), los cuales mantienen una cola de trabajos por cliente. Estos trabajos son representados por ClassAds. La coordinación del matchmaking la realiza el Administrador Central, el cual está compuesto por tres demonios, el *condor_collector*, *condor_accountant* y *condor_negotiator.*

Cada CA envía periódicamente los ClassAds al demonio *condor_collector* del Administrador Central, los cuales contienen información de los usuarios que han emitido trabajos. Los WAs también se encargan de enviar al demonio *condor_collector* del Administrador Central los ClassAds que describen su estado. El demonio *condor_collector* únicamente almacena el ClassAd más reciente de cada WA y de cada cliente emisor de trabajos.

Periódicamente, el demonio *condor_negotiator* del Administrador Central, entra en un ciclo de negociación, durante el cual recupera del demonio *condor_collector* los ClassAds de cada WA y de cada emisor de trabajos. A su vez, solicita al demonio *condor_accountant* que priorice a los emisores teniendo en cuenta el uso que han hecho del recurso en el pasado. Posteriormente, cicla a través de los emisores de trabajos en orden prioritario, y solicita los ClassAds de los trabajos a los CAs. El demonio *condor_negotiator* asocia cada trabajo con un ClassAd de WA compatible (el concepto de compatible está determinado por los requerimientos del trabajo especificados en el ClassAd).

Además de los requerimientos para la ejecución del trabajo, la expresión Rank dentro del ClassAd, permite identificar los recursos preferidos dentro de los requeridos, es decir entre los compatibles.

Cuando el demonio *condor_negotiator* asocia dos ClassAds (de recurso y trabajo), elimina el ClassAd del WA del conjunto de WAs disponibles y envía a cada uno de ellos el ClassAd del otro. El Administrador Central le da al CA un ticket de autorización (el cual fue otorgado por el WA). El CA posteriormente contacta al WA y le envía el ticket de autorización. El WA sólo acepta la solicitud del recurso si el ticket coincide con el que le entregó el colector, y la solicitud coincide con las restricciones establecidas en los ClassAds del trabajo y del recurso.

Si la solicitud es aceptada, el nodo ejecuta el trabajo del cliente e informa al accountan sobre los recursos utilizados. Cuando el CA libera al recurso, libera la solicitud y el WA se anuncia como disponible, enviando un nuevo anuncio al demonio *condor_collector* del Administrador Central.

El WA también envía un anuncio cuando comienza a ejecutar el trabajo, indicando que el nodo está ocupado, pero que está escuchando por solicitudes de clientes de mayor prioridad.

Como ya se ha mencionado, el Administrador Central debe ser altamente tolerante a fallas y el propio demonio *condor_master* se asegura de que los demonios *condor_collector* y *condor_negotiator* siempre estén ejecutándose. Si por alguna razón el demonio *condor_collector* deja de ejecutarse, el demonio *condor_master* inicia uno nuevo, el cual rápidamente deberá aprender de los estados de todos los CAs y WAs, basándose en los mensajes de actualización periódicos. El demonio *condor_negotiator* regenera su lista de ClassAds al comienzo de cada ciclo de negociación. En caso de falla del Administrador Central, las relaciones ya establecidas entre los CAs y WAs no se ven afectadas [36].

4.4 El entorno de ejecución de los trabajos en Condor

Para entender mejor el funcionamiento de Condor a la hora de ejecutar un trabajo, se realiza una comparación de lo que ocurre con las llamadas del sistema si una tarea se ejecuta en un único nodo y lo que ocurre cuando un nodo realiza las llamadas al sistema remoto para ejecutar una tarea.

En la figura 14 se muestra el comportamiento interno de un nodo cuando se ejecuta un programa. El código generado por el usuario interactúa con el kernel del sistema operativo por medio de llamadas internas del mismo con el fin de ejecutar dicha tarea.

En la figura 4 también se muestra cómo trabaja el mecanismo de llamadas al sistema remoto para ejecutar un programa y cómo interactúan dos nodos específicos de un pool de Condor.

Cuando se inicia la ejecución del código del usuario en un nodo, se involucran algunas llamadas al sistema, por ejemplo para el acceso a ficheros, las cuales son interceptadas por el código Condor enlazado al código del usuario de la tarea que se ejecuta (la cual fue enlazada mediante las librerías de Condor, mediante el comando de Condor *condor_compile*). Las librerías de Condor convierten estas llamadas al sistema en llamadas a procedimiento remoto (Remote System Calls) en el nodo emisor de la tarea para que ésta pueda ser ejecutada. La invocación a esa llamada al sistema se realiza por tanto en el nodo emisor del trabajo, y su resultado se devuelve al nodo de ejecución. Por tanto se trabaja de forma transparente, como si la tarea se ejecuta en el nodo de forma local.

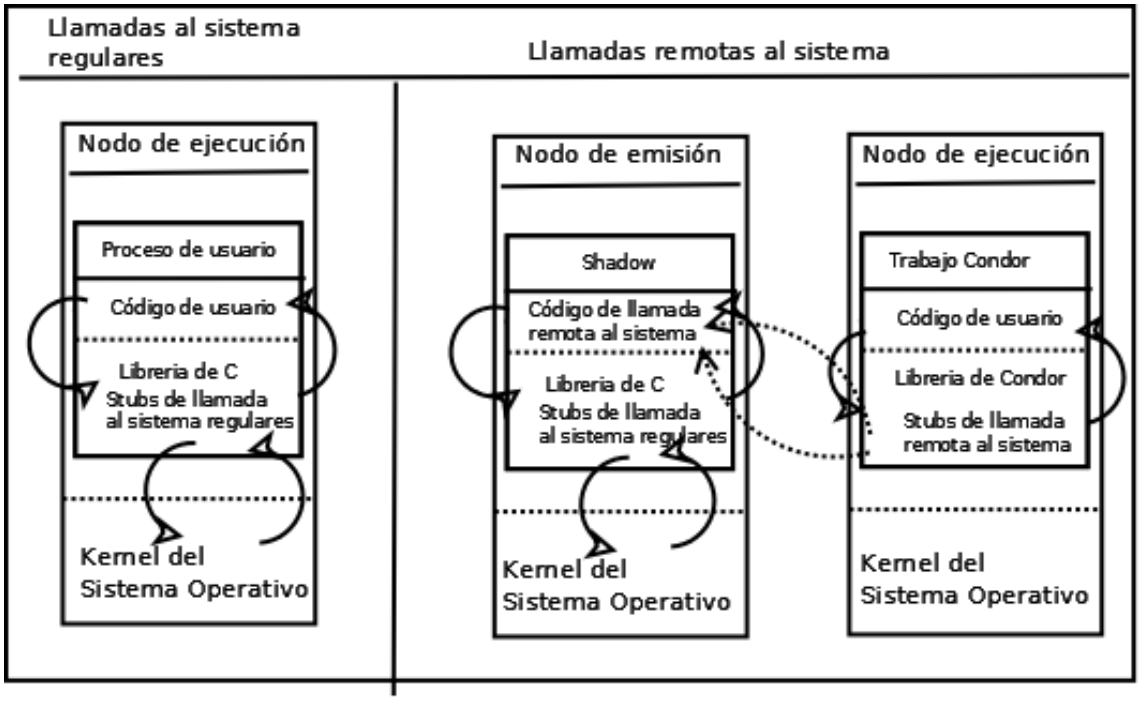

Figura 14. Llamadas al sistema regulares y llamadas remotas al sistema

Es importante destacar que Condor no necesita que los datos estén disponibles en la estación de trabajo remota antes de la ejecución del trabajo, incluso en ausencia de un sistema de archivos compartido.

Condor puede configurarse para ejecutar trabajos en ciertos nodos, sólo cuando el teclado y la CPU estén ociosos. Si el trabajo está ejecutándose en un nodo, regresa el usuario del mismo y presiona una tecla, Condor puede migrar el trabajo a otro puesto y continuar su ejecución desde el punto en el que quedó pendiente.

Este mecanismo también permite planificar en base a prioridades en el pool. Cuando un nodo del pool no está planificado para ejecutar un trabajo, Condor lo utiliza de manera oportunística, pero cuando se necesita ese nodo para ejecutar un determinado trabajo, según lo haya definido el mecanismo de planificación, Condor se apropiará de ese recurso y de cualquier otro que esté ejecutando un trabajo asignado de manera oportunística.

4.4.1 Estado de las máquinas

Es importante conocer el estado en el que se encuentra cada uno de los nodos del pool de Condor así como también las actividades que están realizando en cada momento [37]. Los estados en los que pueden encontrarse los nodos son:

Owner: el estado owner indica que el nodo está en uso en este momento o bien no está disponible para ejecutar los trabajos que Condor solicite. Este es el estado en el que se encuentran los nodos cuando se encienden y la única actividad posible en este estado es Idle.

Idle: Condor considera que un nodo se encuentra Idle cuando el usuario está utilizando el nodo para ejecutar trabajos que no pertenecen a Condor.

Unclaimed: en este estado el nodo está disponible para ejecutar trabajos de Condor, pero no lo está haciendo en este momento. Las actividades posibles en este estado son:

Idle: los nodos en estado unclimed por defecto se encuentran idle. Un nodo que se encuentra en estado unclaimed permite que se ejecuten trabajos de Condor, aunque en este momento no lo está haciendo.

Benchmarking: el nodo está ejecutando benchmarks para determinar su velocidad. Esta actividad solo sucede en el estado unclaimed.

Matched: este estado indica que el nodo está disponible para ejecutar tareas y la única actividad en la que puede encontrarse es idle.

Idle: el nodo se encuentra en condiciones para ejecutar tareas pero todavía no lo está haciendo.

Claimed: cuando el nodo se encuentra en este estado es porque ha sido seleccionado por un demonio *Schedd*. Pueden darse tres actividades diferentes en el estado claimed:

Idle: en esta actividad, el nodo ha sido elegido para ejecutar un trabajo de Condor, pero el demonio *Schedd* que lo seleccionó todavía tiene que activar la selección por medio de una petición al proceso *condor_starter* para atender el trabajo.

Busy: una vez que el demonio *Schedd* se comunica con el proceso *condor_starter* para atender el trabajo, el nodo pasa a la actividad busy, indicando que se encuentra ocupado ejecutando un trabajo.

Suspended: puede suceder que la ejecución del trabajo sea suspendido por Condor, por lo que el nodo cambiará a la actividad suspended. Es importante mencionar que el nodo sigue en estado claimed, pero el trabajo no se está ejecutando momentáneamente, por lo que Condor no devuelve ningún resultado.

Preempting: este estado indica que se ha interrumpido la ejecución de un trabajo de Condor porque el usuario del nodo ha comenzado a utilizarlo.

Vacating: esta actividad indica que el trabajo que se ejecutó está pasando por el proceso de checkpointing. Ni bien se complete este proceso de chequeo, el nodo cambia hacia el estado owner o el estado claimed.

Killing: este estado indica que el nodo ha solicitado que el trabajo que estaba ejecutando libere inmediatamente el nodo sin hacer el proceso de checkpointing.

4.4.2 Gráfico del estado, actividad y transiciones de las máquinas

En la figura 15 se presentan los estados y actividades en los que puede encontrarse un nodo en un pool de Condor. Las flechas indica el orden en el que un nodo puede pasar de un estado a otro.

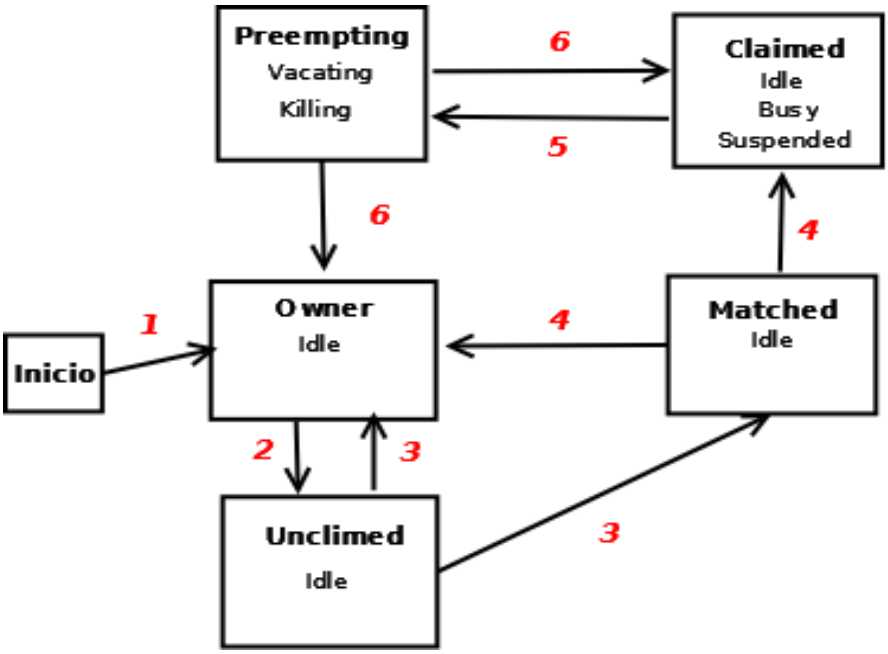

Figura 15. Diagrama de estados, actividades y transiciones de un nodo en un pool de Condor

A continuación se explican las posibles transiciones entre estados.

- Al iniciar Condor en un nodo, su estado inicial es owner y su actividad es idle. Luego de que ha pasado un cierto tiempo de inactividad y dependiendo de cómo este configurado Condor, el nodo pasará al estado unclaimed, notificando al administrador central del pool de Condor que está disponible para ejecutar trabajos.
- En el caso de que el nodo sea ocupado por su usuario, el nodo cambiará al estado owner.
- Si el nodo está disponible para ejecutar trabajos de Condor y mientras el demonio *Schedd* se comunique con el demonio *Negotiator* que reside en el administrador central del pool de Condor, el nodo cambiará su estado y pasará al estado matched.
- Cuando un nodo se encuentra en el estado matched, pueden ocurrir dos cosas:
    1. El nodo puede volver directamente al estado owner porque el usuario del mismo ha comenzado a utilizarlo, razón por la cual ese nodo ya no podrá ejecutar trabajos de Condor.
    2. El nodo va a ejecutar un trabajo de Condor, con lo cual cambia al estado Claimed. A su vez puede encontrarse en:
        1. Idle, debido a que el demonio *Schedd* no ha recibido la petición del proceso *condor_starter* para ejecutar un trabajo,
        2. Busy donde el proceso *condor_starter* ha sido iniciado o
        3. Suspended, donde se ha suspendido el trabajo de Condor.

- Desde el estado claimed, sólo se puede pasar al estado preempting, y este es el estado en el que el trabajo termina. Si el nodo se encuentra en actividad vacating, entonces el trabajo está en proceso de checkpoint o en la actividad killing donde se finaliza el trabajo sin realizar checkpoint.
- En el estado preempting pueden ocurrir dos cosas:

    - Que el nodo vuelva al estado claimed, para terminar la ejecución de algún trabajo que no ha concluido.
    - Que el nodo pase al estado owner donde finaliza el ciclo de ejecución de un trabajo.

En el caso de que los trabajos que se ejecutan sobre Condor no sufran interrupciones por parte de los usuarios de los nodos o cualquier tipo de fallos, las flechas rojas indican el ciclo de estados que atravesaría una máquina durante la ejecución de ese trabajo.

4.5 Envío, ejecución y monitoreo de trabajos en Condor

Para ejecutar un trabajo en el entorno Condor, es necesario llevar a cabo las siguientes tareas:

- Escribir el código para ejecutar el trabajo. Condor ejecuta los trabajos en background, por lo que no interactúa con el usuario. Condor puede redirigir la entrada y la salida estándar desde y hacia un archivo, por tal razón, deben crearse los archivos necesarios y verificar su correcto funcionamiento.
- Elegir el Universo Condor. Condor ofrece diversos entornos de ejecución entre los cuales elegir, dependiendo del tipo de trabajo que se desea ejecutar. En la subsección 4.5.1 se brindan detalles de los universos disponibles en Condor.
- Elaborar el archivo de emisión del trabajo. Todos los detalles del trabajo que va a ejecutarse se encuentran especificados en el archivo de emisión del trabajo, por ejemplo el nombre del archivo ejecutable, los archivos necesarios para que el trabajo pueda ejecutarse, la plataforma sobre la cual se ejecutará el trabajo, etc. En el archivo de emisión del trabajo también se especifica el número de veces que debe ejecutarse el trabajo.
- Emisión del trabajo. Una vez elaborado el archivo de emisión del trabajo, se ejecuta el comando *condor_submit* que toma como parámetro el nombre del archivo de emisión.

Los trabajos se emiten para su ejecución mediante el comando *condor_submit*, el cual toma como argumento el nombre del archivo que desea enviarse. Este archivo debe contener todo lo necesario para que Condor pueda ejecutar el trabajo, como por ejemplo el nombre del archivo ejecutable, el directorio de trabajo donde se encuentra los archivos necesarios, los argumentos del ejecutable, etc. Condor crea un ClassAd del trabajo en base a esta información, para luego proceder a ejecutarlo.

En el archivo de emisión del trabajo, puede especificarse la cantidad de veces que va a ejecutarse el mismo, inclusive si en cada ejecución va a utilizarse un conjunto de datos distintos.

Cada ejecución individual puede tener su directorio de trabajo propio, sus salidas estándar, argumentos y entorno.

En la figura 16 puede observarse un archivo de emisión de trabajo que debe ejecutarse 150 veces y que su archivo ejecutable ha sido compilado y enlazado para un nodo Sun con sistema operativo Solaris 8. En este caso se han establecido requerimientos con la expresión Requirements, como por ejemplo que la memoria RAM sea mayor a 32 MB, y a su vez se establece como preferencia con la expresión Rank que el trabajo se ejecute en nodos con memoria RAM que sea superior a 64 MB, siempre y cuando estén disponibles.

A cada una de las 150 ejecuciones del programa tendrá su propio número de proceso, comenzando desde el 0. Por lo tanto los archivos, files stdin, stdout, y stderr se llamarán in.0, out.0, y err.0 para la primera ejecución del programa, in.1, out.1, y err.1 para la

segunda ejecución del programa, etc. El archivo de log contiene información acerca de cuándo y dónde se ejecuta el programa, checkpointing y registro sobre las migraciones de los procesos. Toda esta información se brinda para cada una de las 150 ejecuciones del programa.

```
Executable = foo
Universe  = standard
Requirements = Memory >= 32 && OpSys == "SOLARIS28" && Arch =="SUN4u"
Rank  = Memory >= 64
Image_Size  = 28 Meg
Error  = err.$(Process)
Input  = in.$(Process)
Output = out.$(Process)
Log   = foo.log
Queue 150
```

Figura 16. Archivo de emisión de un trabajo para ejecutarse múltiples veces

Los requerimientos y preferencias en el archivo de emisión del trabajo deben ser escribirse como expresiones válidas para ClassAds de Condor. En caso de que no se especifiquen, Condor establece valores por defecto. Para indicar los requerimientos y preferencias en el archivo de emisión del trabajo pueden utilizarse los atributos que aparecen en el ClassAd de las máquinas o de los trabajos.

Una vez que se ha emitido el trabajo, el monitoreo de su progreso puede realizarse con los comandos *condor_q* y *condor_status*. Puede modificarse el orden en el que Condor ejecutará los trabajos, mediante el comando *condor_prio.*

Cuando el programa ha concluido su ejecución, Condor informa el estado de la salida del programa con información estadística sobre el rendimiento del programa y tiempo consumido en operaciones de entrada y salida. Es recomendable que toda esta información sobre el resultado de la ejecución del programa quede registrada en un archivo de log.

4.5.1 Los Universos en Condor

Condor presenta diversos universos o entornos de ejecución, siendo el llamado universo Standard el que ofrece el conjunto completo de funcionalidades de Condor, como checkpointing y migración de trabajos, pero solo para programas que pueden ser enlazados con librerías de Condor.

El llamado universo Vanilla brinda menos funcionalidades pero permite la ejecución de una gran variedad de programas, por ejemplo aquellos ficheros binarios que no pueden ser recompilados con librerías de Condor.

Condor ofrece otros entornos de ejecución especializados como Java, Parallel para aplicaciones MPI, Grid para acceder a recursos remotos en una grid, como por ejemplo recursos administrados por Globus Toolkit, etc.

Un universo en Condor define un entorno de ejecución. En la versión de Condor utilizada en esta tesis están disponibles los siguientes universos [38]:

- Standard
- Vanilla
- Grid
- Java
- Scheduler
- Local
- Parallel
- VM

La elección de un universo depende de la naturaleza del trabajo que va a ejecutarse, y se define en el archivo de emisión del trabajo. En caso de que no se defina, el universo por defecto es el Standard.

1. **Universo Standard.** Este entorno utiliza para su funcionamiento el mecanismo de checkpointing y de llamada remota al sistema. Este universo es un entorno de ejecución confiable pero tiene ciertas restricciones respecto a los programas que puede ejecutar.

Los programas que este universo puede ejecutar deben ser compilados con el comando *condor_compile*. Este universo ofrece la posibilidad de realizar checkpointing, lo cual es en esencia una imagen instantánea del estado actual del trabajo. Si el trabajo debe migrar desde un nodo a otro, Condor realiza un checkpoint de la imagen del trabajo, la copia al nuevo nodo y reinicia el trabajo desde su último punto de ejecución. En el caso de que el nodo donde está ejecutándose el trabajo falle o deje de funcionar, Condor puede reiniciar el

trabajo en un nuevo nodo, utilizando la imagen más reciente. De esta manera, un trabajo puede ejecutarse durante largos períodos de tiempo, inclusive si hay fallas en los nodos de ejecución.

Las llamadas remotas al sistema hacen que un trabajo que se ejecuta en uno o varios nodos, perciba que se está ejecutando en su nodo de origen. Cuando un trabajo se ejecuta en un nodo remoto, un proceso llamado condor_shadow se ejecuta en el nodo donde se emitió el trabajo y cuando el trabajo intenta realizar una llamada al sistema, el proceso condor_shadow es el que efectivamente realiza la llamada al sistema y envía el resultado al nodo remoto.

Puede ser necesario convertir el programa que va a ejecutarse a un programa del universo Standard, para lo cual habrá que enlazarlo con las librerías de Condor. No es necesario modificar el código fuente del programa. Es importante mencionar que un programa comercial ya empaquetado como un único ejecutable no puede convertirse en un trabajo del universo Standard. Por ejemplo, si se ha enlazado el trabajo ejecutando:

cc main.o tools.o -o programa

Luego habrá que enlazarlo para Condor ejecutando:

condor_compile cc main.o tools.o -o programa

Hay que tener en cuenta que el nodo desde donde se emiten los trabajos debe tener espacio suficiente para almacenar las imágenes obtenidas del proceso de checkpointig de los trabajos. Estas imágenes son aproximadamente del tamaño de la memoria virtual que consume el trabajo mientras se ejecuta.

2. **Universo Vanilla.** Este universo se utiliza para programas que no pueden ser enlazados con Condor. También se utiliza para ejecutar scripts de shell.

A los trabajos que se ejecutan en el universo Vanilla no puede aplicárseles el mecanismo de checkpointing ni pueden realizar llamadas remotas al sistema. Si la ejecución de un trabajo se interrumpe, Condor puede suspender el trabajo o reiniciar el trabajo desde el principio en otro nodo del pool. Para que un trabajo pueda acceder a archivos de entrada y salida en el universo Vanilla necesitaría utilizar un sistema de archivos compartido como NFS o AFS, ya que no puede realizar llamadas remotas al sistema. Otra alternativa es utilizar el mecanismo de transferencia de archivos que ofrece Condor, mediante el cual Condor transferirá los archivos necesarios al nodo de ejecución y transferirá la salida al nodo emisor del trabajo. En plataformas Linux, Condor asume que se utiliza un sistema de archivos compartidos, pero si no está disponible, el usuario puede habilitar el mecanismo de transferencia de archivos. En el caso de plataformas Windows, por defecto se utiliza el mecanismo de transferencia de archivos. En la figura 17 se observa el archivo de emisión de un trabajo para el universo

Vanilla, que ha sido utilizado en uno de los experimentos computacionales cuyos resultados se discuten en el capítulo 5.

```
universe=vanilla
executable = gaids.run
Rank= (machine == "compute-0-6.local") || (machine == "compute-0-5.local)
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
transfer_output_files = results.output
output=results.output
error=error.output
log=results.log
queue 30
```

Figura 17. Archivo de emisión de un trabajo

En el archivo de emisión del trabajo de la figura 7, se especifica si Condor debe transferir los archivos necesarios para la ejecución del trabajo. La expresión Requirements indica requerimientos específicos y Rank indica preferencias, para que dentro de los recursos que cumplan los requerimientos, el trabajo se ejecute en nodos determinados.

Para posibilitar el mecanismo de transferencia, se colocan dos instrucciones en el archivo de emisión del trabajo: should_transfer_files, la cual especifica que Condor debe transferir archivos desde el nodo que emite el trabajo nodo remoto donde se ejecuta, y when_to_transfer_output que especifica cuándo transferir los resultados.

Así, Condor transfiere los archivos necesarios al nodo, ejecuta el trabajo y transfiere la salida al nodo emisor.

3. **Universo Grid.** El universo Grid ofrece a los usuarios una interfaz estándar para iniciar trabajos en un sistema remoto.

4. **Universo Java.** Un programa emitido para este universo puede ejecutarse sobre cualquier nodo que posea una máquina virtual de Java.

Condor permite que los usuarios accedan a una variedad de recursos distribuidos alrededor del mundo. La Máquina Virtual de Java (JVM) brinda una plataforma

uniforme sobre cualquier máquina, sin tener en cuenta la arquitectura de la misma ni el sistema operativo. El universo Java reúne estas características para crear un entorno distribuido homogéneo.

Los programas compilados en Java pueden ser enviados para ser procesados en Condor, sobre cualquier nodo que pueda ejecutar la Máquina Virtual de Java.

El comando *condor_status –java* puede ser utilizado para visualizar una lista de nodos del pool de los cuales puede disponer Condor. Estos nodos deben ejecutar la Máquina Virtual de Java.

Un número creciente de científicos del ámbito de la computación está considerando cada vez más a Java como un lenguaje adecuado para utilizar en computación distribuida. En el universo Java disponible en Condor, el demonio S*tarter* no ejecuta el trabajo directamente, sino que invoca a la Máquina Virtual de Java, la cual a su vez, invoca el programa Java del usuario. Los archivos de configuración, las librerías y binarios de la Máquina Virtual de Java son especificados por el propietario del nodo y pueden diferir de acuerdo a su ubicación.

El usuario únicamente debe especificar el universo Java en el archivo de emisión del trabajo, y no tiene necesidad de conocer los detalles locales. El demonio *condor_starter* transfiere los archivos de entrada y salida al comienzo y final de la ejecución. Sin embargo, muchos trabajos requieren entradas y salidas intensivas. Para estos programas existe una librería de entrada/salida sencilla, la cual presenta los archivos utilizando abstracciones de Java estándares, tales como las interfaces InputStream y OutputStream. Estas librerías no se comunican directamente con los recursos de almacenamiento, sino que en su lugar invocan a un Proxy mediante un protocolo denominado Chirp [39]. La conexión se establece desde un proceso a otro en la interfaz de loopback de la placa de red. La librería se autentica a sí misma con el demonio *condor_starter*, presentándole una palabra secreta compartida, mediante el sistema de archivos local, logrando una conexión segura.

El Proxy permite al demonio *condor_starter* agregue al trabajo otras funcionalidades adicionales de entrada/salida sin agregar limitaciones al usuario [40].

5. **Universo Scheduler.** Este universo permite que los usuarios ejecuten trabajos livianos y de manera inmediata en el nodo de emisión del trabajo. En este universo Condor no busca un nodo remoto para ejecutar el trabajo. El universo Scheduler, a diferencia del universo Local, no utiliza el demonio *condor_starter* para administrar el trabajo, por lo que sus características son muy limitadas. El universo Local es una mejor opción para la mayoría de los trabajos que se ejecutan en el nodo que lo emite.

6. **Universo Local.** En este universo, el trabajo no espera a ser asociado con algun nodo, sino que empieza a ejecutarse de forma inmediata en el nodo desde el que se emitió.

7. **Universo Paralelo.** Este universo permite la ejecución de trabajos paralelos, como trabajos en MPI. Estos trabajos paralelos se ejecutan de manera oportunística en Condor.

Las aplicaciones MPI pueden procesarse en instalaciones con High Performance Computing, de recursos dedicados, pero no siempre es posible contar con este tipo de recursos y con las instalaciones auxiliares (sistemas de enfriamiento, potencia, etc.). Por lo tanto los recursos parcialmente disponibles resultan ser una valiosa alternativa para muchas instituciones de nuestro país.

Para poder procesar exitosamente aplicaciones MPI en entornos Condor deben resolverse algunos aspectos importantes, como ser la administración de recursos parcialmente disponibles, configuración de MPI (archivos de configuración, instalación de librerias, configuración del protocolo NFS, etc.). Una vez que estos están resueltos estos requisitos, habrá que considerar el overhead que agrega Condor.

En esta tesis, se ejecutaron trabajos MPI [41] sobre Condor, lo cual requiere que se lleven a cabo tareas de configuración previa. En la siguiente lista se indican los pasos necesarios para que Condor funcione con MPI:

1. Instalación de la librería MPI en el nodo maestro y en los nodos esclavos.
2. Intercambio de las claves SSH entre los nodos, ya que MPI necesita acceso a todos los nodos sin que se solicite contraseña.
3. Configuración de NFS en el nodo maestro y en los nodos esclavos, ya que la mayoría de las aplicaciones MPI necesitan leer archivos de entrada y escribir en archivos de salida, desde y hacia un sistema de archivos común.
4. Configuración del nodo de Condor que actúa como Dedicated Scheduler [42]
5. Parametrización del script mp2script [43] de acuerdo al entorno de ejecución a utilizar.
6. Edición del script de bash que establece las variables de entorno e instrucciones necesarias para la ejecución de la aplicación (script.sh).
7. Redacción del ClassAd de Condor.

En el caso de clusters dedicados, una distribución de Linux como Rocks [44] resuelve los tres primeros aspectos de configuración mencionados. Los clusters dedicados utilizados en los experimentos computacionales de MPI sobre Condor que se presentan en el capítulo 5, poseen esta distribución. Consecuentemente el proceso de instalación y configuración es más rápido y sencillo.

En este caso, Rocks ofrece un entorno en el cual no es necesario intercambiar claves entre los nodos, ya que esta tara se realiza durante el proceso de

instalación. También incluye varias librerías MPI, tales como OpenMPI [45] y MPICH2 [46] y permite compartir directorios entre los nodos esclavos mediante NFS.

En el caso de los recursos parcialmente dedicados, no puede instalarse Rocks ni disponer de las facilidades citadas, por lo que todas las tareas deben ejecutarse manualmente.

En el caso de los pools de Condor disponibles para la ejecución de los trabajos MPI que se presentan en el capítulo 5, se utilizó la librería MPICH2 y la instalación en los nodos esclavos se realizó vía NFS, como se indica en el Anexo A. MPICH se instaló en un directorio local del nodo maestro, el cual es exportado vía NFS y montado en cada nodo esclavo.

Cabe destacar que fue necesario intercambiar claves SSH entre los nodos esclavos parcialmente dedicados y el nodo maestro. Para simplificar esta tarea, el directorio home del usuario condor fue exportado vía NFS desde el nodo maestro hacia los nodos esclavos. Todos los nodos esclavos comparten la misma clave SSH y el mismo directorio home. De esta manera se cumple con los ítems 2 y 3 de las tareas indicadas anteriormente.

Al momento de configurar a Condor en el pool, uno de los nodos debe seleccionarse para actuar como Dedicated Scheduler y los trabajos MPI son emitidos desde este scheduler. Cuando el scheduler dedicado solicita un nodo, intentará utilizarlo inmediatamente, y si por alguna razón no puede hacerlo durante un período de tiempo, ese nodo quedará disponible para su uso oportunístico.

Una vez que MPI es invocado desde Condor, los demonios MPI se iniciarán en todos los nodos que han sido elegidos para la ejecución de trabajos. Posteriormente, Condor no se involucra en la ejecución de trabajos MPI, ya que el middleware espera hasta que los trabajos terminen y devuelvan el control. En ese momento, un archivo de Condor reporta el tiempo total de ejecución, e indica los errores si por alguna razón la ejecución no ha sido exitosa.

Cada recurso es controlado por el nodo Dedicated Scheduler y debe tener su nombre incluido en el archivo de configuración local [47].

En definitiva, Condor se involucra en el momento del matchmaking o asociación, cuando el trabajo es enviado para su ejecución y al final del proceso, cuando hayan concluido su trabajo los demonios MPI En este momento se envía una notificación al usuario.

Al momento de configurar el archivo de emisión del trabajo de Condor, se especifica el archivo ejecutable, que en este caso es el script denominado mp2script. Este script contiene la ruta desde la cual son lanzados los demonios MPI en cada nodo y el número de demonios MPI a ser lanzados. Este script recibe de Condor la lista de nodos que ejecutarán el trabajo paralelo y se encarga también de ejecutar el demonio MPI en cada nodo elegido por Condor.

Una vez que el trabajo ha concluido normal o anormalmente, el script notifica a Condor y genera las salidas correspondientes.

Dependiendo del tipo de aplicación paralela que vaya a ejecutarse, deberá configurarse otro script denominado script.sh, el cual se envía como parte de los argumentos. Este archivo contiene algunas variables de entorno que especifican algunas tareas que son condiciones necesarias para que se ejecuten los trabajos. Este script lleva a cabo la ejecución de la aplicación. En la figura 18 se muestra el script.

```
#!/bin/bash -x
EXECDIR="/home/condor/mc2tst"
export rep_from_which_model_is_launched=‘pwd‘
export AFSISIO=$EXECDIR/./mc2/data/
export F_UFMTENDIAN=big
ulimit -s unlimited
cd $EXECDIR
mkdir $EXECDIR/tmp
export TMPDIR=${EXECDIR}/tmp
$EXECDIR/mc2dm.Abs > $EXECDIR/salida_mc2.out
```

Figura 18. Ejemplo de script.sh

La ventaja de la utilización de Condor para ejecutar trabajos MPI en recursos parcialmente dedicados reside en el hecho de que no es necesario configurar un archivo con el listado de todos los recursos disponibles (los cuales pueden o no ser dedicados). Si los trabajos son lanzados con MPI nativo, este archivo debe configurarse manualmente, el cual comúnmente se llama machinefile o mpd.hosts. Este archivo contiene los nombres de los nodos y número de procesos MPI que van a ejecutarse por nodo.

Además de configurar el archivo con los nombres de los nodos que ejecutarán el trabajo, Condor permite especificar cualquier restricción o requerimientos que los nodos deben cumplir para ejecutar el trabajo. Por ejemplo, puede requerirse que el nodo tenga un load avarage específico, una cantidad mínima de memoria RAM o una cantidad mínima de espacio libre de disco.

Estos requerimientos no pueden especificarse cuando se utiliza sólo MPI, sin Condor y otro scheduler como PBS [7], LFS [6], Torque [9], etc. Para mayor

información se pueden consultar los comandos mpiexec [48]. Condor enviará trabajos solo a esos nodos, dejando a los demás para uso oportunístico.

La figura 19 es un ejemplo de un archivo de emisión de un trabajo de Condor (ClassAd) para la emisión de un trabajo paralelo. Este ClassAd fue configurado para emitir la aplicación del modelo meteorológico Mesoscale Compressible community model (MC2) [49] la cual se presentará en el capítulo 5.

```
Requirements= (machine == labredes01.labredes.ryt.itu.uncu.edu.ar)||
 machine == labredes02.labredes.ryt.itu.uncu.edu.ar)||
 machine == labredes03.labredes.ryt.itu.uncu.edu.ar)||
 machine == labredes04.labredes.ryt.itu.uncu.edu.ar)||
 machine == labredes05labredes.ryt.itu.uncu.edu.ar)||
 machine == labredes06.labredes.ryt.itu.uncu.edu.ar)||
 machine == labredes07.labredes.ryt.itu.uncu.edu.ar)||
 machine == labredes08.labredes.ryt.itu.uncu.edu.ar)||
 machine == labredes09.labredes.ryt.itu.uncu.edu.ar))
 log = logfile
 output = outfile.$(NODE)
 error = errfile.$(NODE)
 arguments = script.sh
 machine_count = 8
 should_transfer_files = yes
 when_to_transfer_output = on_exit
 queue
```

Figura 19. ClassAd de Condor

8. **Universo VM.** Condor facilita la ejecución de máquinas virtuales como Vmware y Xen, dentro de este universo.

4.5.2 Emisión de trabajos utilizando un sistema de archivos compartidos

En el caso de que no se utilice el mecanismo de transferencia de archivos para emitir trabajos, Condor deberá utilizar un sistema de archivos compartidos para acceder a los archivos de entrada y salida que necesite el trabajo para su ejecución.

Los usuarios pueden asegurarse que sus trabajos tengan acceso al sistema de archivos correcto mediante los atributos de ClassAd de nodos FileSystemDomain y UidDomain. Estos atributos indican los nodos tienen acceso al mismo sistema de archivos compartido.

Se considera que todos los nodos que montan los mismos directorios compartidos en la misma ubicación pertenecen al mismo sistema de archivos compartidos. Similarmente, todos los nodos que comparten la misma información de usuario, es decir el mismo UID, lo cual es importante particularmente para NFS, son considerados parte del mismo dominio.

La configuración por defecto de Condor establece que cada nodo tenga su UID de dominio y de sistema de archivos propio, utilizando el nombre completo del nodo como nombre de los dominios. Entonces si un pool tiene acceso a un sistema de archivos compartidos, Condor debe configurarse para que todos los nodos que monten los mismos archivos tengan la variable FileSystemDomain correctamente configurada, con el mismo valor.

Cuando un trabajo tiene que ser ejecutado en un sistema de archivos compartido, la expresión requirements del archivo de emisión del trabajo es utilizada por Condor para asegurarse que el trabajo se ejecute en un nodo con los valores de UidDomain y FileSystemDomain correctos. En el caso que no se definan valores para estas variables a la hora de emitir el trabajo, éste se ejecutará en un nodo con el mismo UidDomain y FileSystemDomain que el del nodo que lo emitió.

Es importante mencionar que si no está configurado un sistema de archivos compartido o no se configura correctamente la variable FileSystemDomain y el usuario emite un trabajo que no puede ejecutar llamadas remotas al sistema y no está configurado el mecanismo de transferencia de archivos, el trabajo se ejecutará sólo en el nodo desde el que fue emitido.

4.5.3 Emisión de trabajos sin utilizar un sistema de archivos compartidos

Condor puede funcionar sin utilizar un sistema de archivos compartido, implementando el mecanismo de transferencia de archivos de Condor, el cual es utilizado por el usuario al momento de emitir el trabajo. Condor transferirá todos los archivos necesarios para el trabajo desde el nodo del cual se emitió el trabajo. Estos archivos se almacenan en un directorio temporal en el nodo donde se ejecutará el trabajo. Una vez que Condor ejecuta el trabajo, transfiere la salida al nodo que emitió el trabajo.

El usuario especifica los archivos que deben transferirse y en qué punto los archivos de salida deben ser copiados al nodo emisor. Esto se indica en el archivo de emisión del trabajo.

Cuando se emiten trabajos en el universo Standard, no es necesario contar con un sistema de archivos compartido, ya que el acceso a los archivos se realiza mediante el mecanismo de llamadas remotas al sistema de Condor. Mediante este mecanismo, el archivo ejecutable y los archivos de checkpoint se transfieren automáticamente cuando es necesario. De esta manera, el usuario no necesita modificar el archivo de emisión del trabajo en caso de no contar con un sistema de archivos compartido.

Para los universos Vanilla, Java y Parallel, en caso de no contar con sistema de archivos compartido, debe explícitamente habilitarse el mecanismo de transferencia de archivos de Condor.

Para el caso particular de emitir trabajos desde un nodo con sistema operativo Windows, Condor asume que no tiene acceso a un sistema de archivos compartido. En este caso habilita por defecto el mecanismo de transferencia de archivos y al momento de emitir el trabajo se debe especificar cuáles son los archivos a transferir y cuándo se deben transferir los archivos de salida.

Para el universo Grid, en el cual los trabajos se ejecutan en nodos remotos, no debería haber un sistema de archivos compartidos entre nodos.

Para el universo Scheduler, Condor utiliza como nodo de ejecución el nodo desde el cual fue emitido el trabajo, por lo tanto no es necesario disponer de un sistema de archivos compartido [50].

4.5.4 Monitoreo de los trabajos en Condor

Una vez que los trabajos son emitidos, Condor buscará los recursos necesarios par su ejecución. El comando *condor_satutus –submitters* permite visualizar todos los nodos que han emitido trabajos.

El comando *condor_q* permite observar el estado en el que se encuentran los trabajos en cada instante. Los trabajos pueden encontrarse en estado de ejecución, que se indica con una letra R (running), en estado ocioso, que se indica con una letra I (idle). Este estado indica que el trabajo no se está ejecutando porque está esperando un nodo disponible.

Los trabajos pueden estar en estado detenido, que se indica con una letra H (hold). En este caso el trabajo no será planificado para ejecutarse hasta ser liberado.

Otra forma de visualizar el estado de los trabajos es mediante el archivo de log, lo cual debe especificarse en el archivo de emisión del trabajo.

Cuando un trabajo comienza a ejecutarse, Condor inicia el proceso condor_shadow en el nodo que emitió el trabajo. Este proceso permite que los trabajos que están ejecutando remotamente accedan al entorno desde el cual fueron emitidos y a los archivos de entrada y salida. Si un nodo ha emitido cientos de trabajos, tendrá cientos de procesos condor_shadow ejecutándose en el nodo.

Los trabajos pueden ser removidos de una cola mediante el comando *condor_rm*. En el caso que el trabajo removido esté actualmente en ejecución, será eliminado sin realizar el proceso de checkpointing.

Los trabajos pueden suspenderse mediante el comando *condor_hold*. Un trabajo en el estado suspendido permanece en ese estado hasta que sea liberado para continuar su ejecución mediante el comando *condor_release*. Para los trabajos que se ejecutan en el universo Standard no se realiza checkpointing antes de ser suspendidos. Una vez que son liberados para continuar ejecutándose, lo harán desde el punto de checkpointing disponible más reciente. Para los demás universos, los trabajos suspendidos comenzarán desde el principio.

Además de las prioridades que posea cada usuario del sistema, Condor permite que cada usuario asigne prioridades a los trabajos que envía. Estas prioridades tienen carácter local a cada cola de trabajos, y debe estar representada por un valor entero, el cual cuanto más grande sea indica mayor prioridad. Por defecto, la prioridad de un trabajo es 0 y puede modificarse con el comando *condor_prio.*

Es importante destacar que las prioridades de los trabajos son completamente diferentes a las prioridades asignadas por Condor a los usuarios. Las prioridades de los trabajos no tienen impacto en las de los usuarios, ya que sólo son un mecanismo para que el usuario identifique la importancia relativa de los trabajos entre todos los trabajos emitidos por el usuario a una cola específica.

Puede ocurrir que los trabajos no se ejecuten. A continuación se presentan algunas razones.

El comando *condor_q* permite visualizar los trabajos que no están ejecutándose. Una de las razones más habituales por las que los trabajos no se ejecutan, es que Condor no ha completado el ciclo de negociación, en el cual los trabajos encolados son asignados a los nodos del pool para que comiencen su ejecución. Este evento ocurre por defecto una vez cada cinco minutos, por lo que el usuario debería esperar por lo menos cinco minutos para detectar las razones por las que un trabajo no se ejecuta. Otros problemas pueden ser diagnosticados con el comando *condor_q –analyze*.

Otros tipos de problemas lo presentan los trabajos que se ejecutan o ejecutaron por un período corto de tiempo y han dejado de hacerlo. En este caso el comando *condor_q* puede no ser suficiente ya que solo informa sobre el estado actual del trabajo. La información que permite detectar el problema se puede encontrar en el archivo de log o de error que se especifican en el archivo de emisión del trabajo.

Los trabajos pueden ejecutarse por un período de tiempo corto y luego terminarse por problemas de permisos en los archivos. Para este tipo de problemas, el archivo de log que utiliza el demonio *condor_shadow* contendrá información acerca de este problema. Este archivo de log está asociado con el nodo desde el cual se emitió el trabajo.

Otra causa de problemas suelen ser los nodos con abundante memoria física pero muy poco o nulo espacio para swap. Condor no tiene en consideración el tamaño de la memoria física, por lo tanto los problemas aparecen cuando Condor supone que no tiene espacio swap para trabajar, por lo que no ejecutará los trabajos emitidos.

En los archivos de log puede encontrarse de manera cronológica todos los eventos que ocurren durante el ciclo de vida de un trabajo. El formato de los eventos que se detallan

en el archivo de log incluye cuatro campos. El primer campo es un valor numérico que indica el tipo de evento. El segundo campo identifica al trabajo que generó el evento y se detallan algunos atributos del ClassAd. El tercer campo indica la fecha y la hora a la que ocurrió el evento. Finalmente el cuarto campo describe brevemente el evento.

Cuando el trabajo termina su ejecución, Condor lo elimina de la cola de trabajos y lo agrega al archivo historial del trabajo. El historial del trabajo puede verse con el comando *condor_history.* Al concluir la ejecución de un trabajo, Condor envía un mail con el estado de salida del trabajo [51].

4.6 Prioridades en Condor

4.6.1 Prioridades de los trabajos

Las prioridades de los trabajos permiten asignar un nivel de prioridad a cada trabajo de Condor enviado para ejecución, con el objetivo de controlar el orden de ejecución. En la figura 20 se observa la implementación del comando *condor_prio* para modificar la prioridad de un trabajo. La prioridad de un trabajo puede tomar cualquier valor entero, y cuanto mayor sea, mayor prioridad tendrá en la cola de trabajos. Como se mencionó anteriormente, por defecto, la prioridad de un trabajo es 0.

```
%  condor_q

-- Submitter: tesla.itu.uncu.edu.ar : <x.x.x.x:x> : tesla.itu.uncu.edu.ar

ID      OWNER         SUBMITTED    CPU_USAGE ST PRI SIZE  CMD

100.0   condor        4/09  11:10  0+00:00:00 I  0   0.3   gaids.run

1 jobs; 1 idle, 0 running, 0 held

%  condor_prio -p -15 100.0

%  condor_q

-- Submitter: tesla.itu.uncu.edu.ar : <x.x.x.x:x> : tesla.itu.uncu.edu.ar

ID      OWNER           SUBMITTED    CPU_USAGE ST  PRI  SIZE CMD

100.0   condor          4/09  11:10  0+00:00:00 I  -15  0.3  gaids.run

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

Figura 20. Cambio de prioridad de un trabajo

4.6.2 Prioridades de los usuarios

Los recursos se asignan a los usuarios teniendo en cuenta la prioridad del usuario. Cuanto más bajo sea el valor numérico, mayor es la prioridad del usuario. Por lo tanto un usuario con prioridad 5 obtendrá más recursos que un usuario con prioridad 50.

Las prioridades en Condor pueden ser examinadas con el comando *condor_userprio*, y el administrador de Condor puede configurar y modificar la prioridad de un usuario en particular con el mismo comando.

Condor calcula continuamente la proporción de recursos disponibles que se debería asignar a cada usuario. Esta proporción está inversamente relacionada con la prioridad del usuario. Por ejemplo, un usuario con prioridad de 10, obtendrá el doble de recursos que un usuario con prioridad de 20.

La prioridad de cada usuario se modifica de acuerdo al número de recursos que cada usuario está utilizando. Cada usuario comienza con la mejor prioridad posible que es 0.5. Luego si el número de recursos que está utilizando actualmente es mayor que su prioridad, la prioridad del usuario disminuirá a lo largo del tiempo. Si el número de recursos es menor que la prioridad, la prioridad aumentará a lo largo del tiempo. La idea final es que todos los usuarios tengan acceso equitativo a los recursos para ejecutar sus trabajos.

Por defecto, Condor ajusta los valores de las prioridades cada 24 horas, pero este valor se puede modificar.

Además de asegurarse que cada usuario reciba de manera justa los recursos necesarios para ejecutar trabajos, Condor se apropiará de los recursos que posea un usuario de baja prioridad si éstos son reclamados por un usuario de mayor prioridad. En este caso Condor realizará un checkpointing de los trabajos del usuario de menor prioridad y los dejará en estado vacante. Así quedarán libres los recursos que Condor asignará al usuario de mayor prioridad.

Sin embargo, Condor sólo dejará vacantes los trabajos del usuario de menor prioridad que permitan iniciar los trabajos de usuarios de mayor prioridad. Por defecto, Condor dejará vacantes los trabajos de usuarios de menor prioridad que se hayan ejecutado por lo menos durante una hora.

Las prioridades de los usuarios se basan en el formato nombre_usuario@dominio, por lo tanto la prioridad del usuario y la asignación de recursos no se ve afectada por el nodo que utilice el usuario para emitir los trabajos.

Otra opción que ofrece Condor en cuanto a prioridades, es la posibilidad de enviar un trabajo como nice. Esto quiere decir que el trabajo sólo se ejecutará en nodos que ningún otro trabajo Condor requiera o solicite [52].

4.7 El mecanismo de checkpointing en Condor

Efectuar checkpointing de un trabajo es similar a tomar una instantánea del estado actual del trabajo de tal forma que pueda ser reiniciado posteriormente desde ese punto. Si el planificador de Condor decide no continuar asignando un nodo a un trabajo, por ejemplo porque ha regresado el propietario del mismo, puede realizar el checkpointing del trabajo para luego continuar su ejecución desde este punto y no perder lo realizado hasta ese momento. El trabajo podrá reanudarse luego en otro nodo que le sea asignado.

Condor realiza también un proceso de checkpointing periódico, lo cual es un mecanismo para implementar tolerancia a fallos. Las "instantáneas" son tomadas de forma periódica y luego de que se ha producido una interrupción en el servicio, el trabajo continúa desde esta última instantánea. Cabe destacar que el servicio de checkpointing de Condor es opcional.

Los procesos de Condor para los cuales está habilitado el mecanismo de checkpointing, son sometidos a este mecanismo cuando son retirados del nodo en el cual se estaban ejecutando. Al momento de encontrar un nodo adecuado para retomar su ejecución (con la misma arquitectura y sistema operativo), se retoma la ejecución del proceso desde el último punto de checkpointing y a los trabajos a los cuales no se les efectuó el checkpointing se los reinicia desde el principio.

Una alternativa para no tener que reiniciar el trabajo desde el principio es freezar el trabajo mientras se ejecuta otra tarea, y luego desfreezarlo para que pueda continuar donde quedó. Esto no requiere ningún tratamiento especial del trabajo, a diferencia de otras estrategias que implementan mecanismos de checkpoint, pero sí requiere una configuración especial de Condor.

En los nodos del pool de Condor se configura una expresión que controla cuándo y qué tan seguido se realiza el checkpointing periódico, por ejemplo nunca, cada tres horas, etc. Al momento de realizarse el checkpointing, se suspende la ejecución del trabajo, se realiza el checkpointing e inmediatamente continúa su ejecución. Por otro lado el usuario puede solicitar que se efectúe el checkpointing del trabajo con el comando *condor_ckpt.*

En el caso de que se produzca un error durante el proceso de checkpointing, el trabajo se reanudará desde el punto de checkpointing previo. Cuando se realiza un nuevo checkpoint de un trabajo se elimina el checkpoint previo, siempre y cuando se haya efectuado exitosamente. Por defecto, el checkpoint de un proceso se escribe en el disco local del nodo desde el cual se emitió el trabajo. También puede configurarse un servidor de checkpointing, para no utilizar el disco del nodo desde el cual se emitió el trabajo [53].

4.8 Workflows

Las aplicaciones a gran escala, en la forma de workflows, pueden necesitar la utilización coordinada de recursos que pueden estar esparcidos a través de distintos dominios administrativos.

Un workflow consiste en la ejecución de numerosas aplicaciones interdependientes (probablemente diferentes). Por ejemplo, una aplicación que debe generar datos iniciales, que luego se deben pasar a otra aplicación, la cual a su vez debe pasar a otra aplicación que ejecuta simulaciones. Al concluir todas las simulaciones, otra aplicación puede recoger estos datos y realizar otro análisis sobre esos datos.

Como se observa, puede haber interdependencias en distintos niveles de un mismo proceso y algunas aplicaciones pueden tener que esperar por otras a que terminen antes de continuar. Condor provee un motor de workflow denominado DAGMan que se encarga de administrar este tipo de situaciones [54].

Dentro de Condor, DAGMan es ejecutado como un trabajo del universo Scheduler. A medida que DAGMan envía programas para su ejecución, realiza un monitoreo de los mismos, que se ven reflejados en archivos de log, a fin de asegurarse el orden requerido dentro del DAG. DAGMan es responsable además de la planificación, recupero y reporte de los programas enviados a Condor.

4.8.1 La terminología DAGMan

Un nodo en DAGMan puede abarcar más que un simple programa enviado por Condor. En un principio, el número de trabajos de Condor por nodo, estaba restringido a uno, pero ahora esta restricción ya no es tal, porque los trabajos de Condor por nodo comparten un único número de cluster. La única limitación que existe es que todos los trabajos de un cluster deben utilizar el mismo archivo de log. Distintos nodos dentro de un DAG pueden utilizar distintos archivos de log.

A medida que DAGMan planifica y envía trabajos a los nodos de Condor para su ejecución, estos trabajos se van definiendo como exitosos o no teniendo en cuenta los valores de retorno. El resultado exitoso o no es propagado de formas bien definidas en los nodos que componen un DAG. La continuidad de la ejecución de las tareas depende del éxito o falla en uno o más nodos.

Es importante destacar que la falla de un trabajo que está ejecutándose en un cluster de múltiples trabajos, dentro de un nodo, provoca que el cluster completo falle. Cualquier otro trabajo que esté ejecutándose dentro del cluster de trabajos que ha fallado es removido inmediatamente.

Los trabajos en DAGMan son enviados mediante el comando *condor_schedd*. Este comando lanza el trabajo DAG dentro del universo Scheduler.

4.8.2 Envío y monitoreo de trabajos DAG

Los archivos de emisión de los trabajos DAGMan actúan como punteros para los archivos de emisión de trabajos individuales, e indican el orden en que deben ejecutarse los trabajos. En el archivo de emisión del trabajo DAGMan, las dependencias de los

trabajos están expresadas mediante relaciones PARENT-CHILD. Usualmente, la configuración básica de los trabajos de DAGMan se parece a un diamante, como se muestra en la figura 11. En la figura 21, se observa que el trabajo N1 debe completarse exitosamente para que N2 y N3 se puedan ejecutar en paralelo. Solamente cuando ambos han concluido exitosamente, N4 comenzará su ejecución. En este ejemplo, N5 está desconectado del nodo, por lo que puede ejecutarse en paralelo con los trabajos de otros nodos [55].

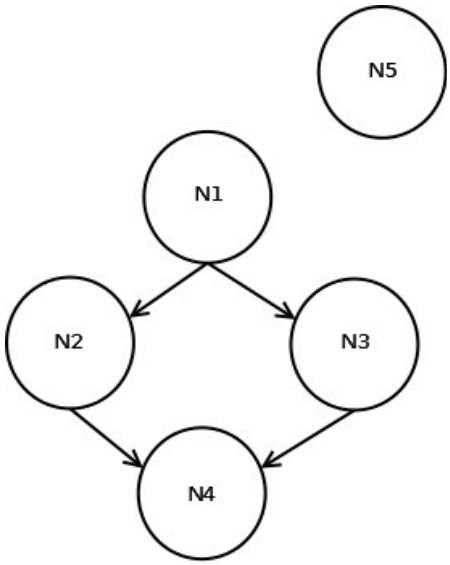

Figura 21. Enlaces tipo parent-child

En la figura 22 se observa la sintaxis de un archivo de emisión de un trabajo DAG.

```
# this file is called diamond.dag
JOB A A_job.submit
JOB B B_job.submit
JOB C C_job.submit
JOB D D_job.submit
PARENT A CHILD B,C
PARENT B,C CHILD D
```

Figura 22. Archivo de emisión de un trabajo DAG

A la hora de emitir un trabajo DAG, deben seguirse los siguientes pasos:

- Definir los trabajos. Cada uno debe tener un nombre y una ubicación. La ubicación es el archivo de emisión para el trabajo individual.
- Definir los pares padres/hijos. Los trabajos padres son aquellos que deben ejecutarse primero, ya que los trabajos hijos no pueden comenzarse a ejecutar hasta que los trabajos padres hayan terminado exitosamente.
- En el archivo de emisión del trabajo no es necesario explicitar que va a utilizarse el universo Scheduler.
- Cuando los archivos de emisión de los trabajos están completos, deben salvarse en un archivo para luego poder emitirlos con el comando *condor_submit_dag*. Desde el promt el shell, el comando *condor_submit_dag* permite enviar el trabajo, seguido del nombre del archivo de emisión:

$ condor_submit_dag diamond_dag

- Este comando mostrará el siguiente resultado en la consola:

```
Checking all your submit files for log file names.
This might take a while...
Done.
------------------------------------------------------------------------
File for submitting this DAG to Condor          : diamond_dag.condor.sub
Log of DAGMan debugging messages                : diamond_dag.dagman.out
Log of Condor library output                    : diamond_dag.lib.out
Log of Condor library error messages            : diamond_dag.lib.err
Log of the life of condor_dagman itself         : diamond_dag.dagman.log

Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 30072.
------------------------------------------------------------------------
```

Si el trabajo se ha enviado exitosamente, el comando *condor_submit_dag* brinda un resumen del resultado del envío del trabajo y la ubicación de los archivos de log. El archivo de log más relevante es el denominado lifetime, ya que es utilizado para coordinar la ejecución del trabajo.

- El trabajo DAG se considera inválido si incurre en ciclos, los cuales se detectan durante la ejecución de la rutina de control de consistencia, y en caso de detectar un bucle, se generará el siguiente mensaje de error:

ERROR: a cycle exists in the DAG

- Es posible enviar más de dos DAG al mismo tiempo, listando los archivos a emitir:

$ condor_submit_dag dag_file1, dag_file2, dag_file3

- Los trabajos DAG pueden monitorearse con el comando *condor_q,* y para visualizar mayor cantidad de información acerca de los trabajos DAG, puede utilizarse la opción –dag siguiendo al comando *condor_q.*
- Para eliminar un trabajo DAG, se utiliza el comando *condor_rm* indicando el número de trabajo. Una vez que el trabajo DAG es removido, todos los trabajos asociados serán removidos también [56].

Capítulo 5

5. Experimentos

En este capítulo se presentan los experimentos computacionales ejecutados sobre el middleware Condor.

El objetivo de demostrar cómo Condor explota y administra de manera eficiente los recursos disponibles en un entorno de Computación de Alta Disponibilidad.

En la sección 5.1 se presenta una aplicación que permite obtener patrones de tráfico de red, la cual se ejecuta sobre una infraestructura Condor, obteniéndose una importante reducción en el tiempo de ejecución necesario para la aplicación.

En la sección 5.2 se presenta una aplicación para llevar a cabo estudios paramétricos de mecánica de sólidos en entornos de computación distribuida. Como se discute en la subsección 5.2.13 La ejecución de esta aplicación en un entorno como Condor permite ahorrar de tiempo de procesamiento y de análisis en el caso de estudios paramétricos.

En la sección 5.3 se presenta el procesamiento de aplicaciones MPI sobre recursos parcialmente dedicados administrados por Condor. En primera medida, se implementa la multiplicación de matrices, la cual es una operación es muy adecuada para paralelización. Posteriormente se discute y presentan los resultados de la ejecución del modelo meteorológico Canadian Mesoscale Compressible Community Model (MC2), sobre una infraestructura administrada por Condor, el cual no agrega un overhead importante.

5.1 Reconocimiento de patrones de tráfico de red en un ambiente Condor

Se presenta a continuación la ejecución en un ambiente Condor de una aplicación diseñada para obtener patrones de tráfico de red. En la referencia [57] se resumieron los resultados y aspectos más destacados del problema que se discute en esta sección.

La ejecución de la aplicación sobre el ambiente Condor no sólo ha permitido mejorar el tiempo total de ejecución sino también obtener mejores resultados respecto del caso serial. Pese a la pequeña infraestructura utilizada en esta experiencia, las mejoras obtenidas muestran la capacidad de Condor como un sistema por lotes de computación distribuida de alta disponibilidad.

Cabe destacar que fue necesario realizar cambios para adaptar al entorno Condor la aplicación de reconocimiento de patrones en el tráfico de red basada en algoritmos genéticos. Estos cambios se discuten en la subsección 5.1.4 [58].

Debido a la característica no determinística de las aplicaciones basadas en algoritmos genéticos, surge la necesidad de repetir su ejecución numerosas veces con el fin de realizar las validaciones estadísticas. La característica fuertemente desacoplada de este

tipo de validaciones estadísticas y el alto consumo de recursos computacionales que las mismas demandan, sugieren su ejecución en ambientes distribuidos como Condor.

Para ejecutar los experimentos computacionales de la aplicación, se utilizaron las infraestructuras presentadas en las figuras 2 y 3 de la sección 2.3 del capítulo 2 de esta tesis.

A fin de procesar la aplicación para obtener patrones de tráfico de red, es necesario crear los archivos que contienen las entradas correspondientes para el programa a ejecutar, el cual se muestra en la figura 23.

```
universe=vanilla

executable = gaids.run

Rank= (machine == "compute-0-6.local") || (machine == "compute-0-
5.local)

should_transfer_files = YES

when_to_transfer_output = ON_EXIT

transfer_output_files = results.output

output=results.output

error=error.output

log=results.log

queue 30
```

Figura 23. Archivo de emisión de un trabajo

En el archivo de emisión del trabajo que se muestra en la figura 23, se especifica que Condor debe transferir los archivos necesarios para la ejecución del trabajo. La expresión Rank indica preferencias, que en este caso indican que el trabajo se ejecute en determinados nodos.

Por otro lado, para posibilitar el mecanismo de transferencia, se colocan dos instrucciones en el archivo de emisión del trabajo: should_transfer_files, la cual especifica que Condor debe transferir archivos desde el nodo que emite el trabajo nodo remoto donde se ejecuta, y when_to_transfer_output que especifica cuándo transferir los resultados.

De esta forma, Condor transfiere los archivos al nodo, lo ejecuta y transfiere la salida al nodo que emitió el trabajo.

5.1.1 Introducción al problema del reconocimiento de patrones en el tráfico de red

Las falencias en la seguridad de protocolos como ARP, TCP, TELNET, SMTP, FTP han sido la causa de ataques contra la confidencialidad, la disponibilidad y la autenticidad de los datos transportados. Si bien estos problemas han sido corregidos a lo largo de los años, continuamente se van descubriendo nuevas maneras de realizar estos ataques.

El ingeniero en seguridad de redes debe estar alerta para detectar estos ataques, informándose de las nuevas vulnerabilidades descubiertas o tipos de ataques perpetrados. Sin embargo una gran cantidad de estos ataques tienen lugar antes que se conozcan siquiera las vulnerabilidades o fallas que los provoca.

Para hacer frente a esto es que en los últimos años han surgido propuestas para la aplicación de técnicas de inteligencia artificial en el ámbito de la seguridad en redes [59, 60, 61].

Con este objetivo se utiliza un algoritmo genético para el reconocimiento de patrones en el tráfico de red como punto de partida para abordar un problema de mayor envergadura como lo es la detección de intrusos por anomalías en el tráfico de red. Entendiéndose por una anomalía a toda instancia de tráfico que se aparte del comportamiento normal de la red [62].

5.1.2 Algoritmo genético propuesto

El algoritmo genético propuesto parte de una población de individuos conformados por instancias de tráfico de red elegidas al azar, para al finalizar obtener el conjunto de reglas que más coincidencias encuentre en el tráfico de la red.

Para la representación de la población se seleccionan 6 atributos de una instancia de tráfico: tiempo de duración de la conexión, tipo de protocolo, puerto origen, puerto destino, dirección IP origen y dirección IP destino.

Se propone una función de fitness definida en la ecuación (1) que favorece a aquellos individuos de la población que presenten mayor cantidad de coincidencias en los atributos de las instancias de tráfico.

$$f(r) = \frac{(\prod_{j=1}^{j=m} \prod_{i=1}^{i=n} \alpha(ri,dij))}{|D|} \qquad (1)$$

Para calcular el valor de fitness del individuo r, se comparan los genes del individuo r con los correspondientes d_{ij} de cada una de las instancias de tráfico pertenecientes al conjunto de entrenamiento D mediante la función (2) definida como:

$$\alpha(r_j, d_{ji}) = \begin{cases} \omega'_i & \text{si } r_j \neq d_{ij} \\ \omega_i & \text{si } r_j = d_{ij} \end{cases} \quad (2)$$

Donde cada gen tiene un peso asociado ωi con el objeto de favorecer a aquellos individuos que por experiencia disciplinar resultan más relevantes. Cuando algún gen no presente coincidencia, se le asigna un peso ω´ij, que se ajusta en la práctica al valor 0,1.

De esta manera la función de fitness favorece a los genes que presentan una frecuencia de aparición relativamente alta. Estos individuos en futuras generaciones, pueden originar nuevas reglas que coincidan con un significativo número de instancias de tráfico.

Con el fin de comprobar el funcionamiento del algoritmo propuesto se realizan experimentos que utilizan 2 conjuntos conformados con instancias de tráfico tomadas del conjunto de datos provisto por DARPA [63]. El primer conjunto contiene 9000 entradas que representan 4 horas de tráfico, utilizado para la etapa de entrenamiento. El segundo conjunto contiene 35.000 instancias de tráfico que representan 24 horas, utilizado para la fase de prueba.

El algoritmo necesita de un ajuste en sus parámetros con el fin de encontrar la combinación que presente mejores resultados.

Sin embargo debido a la característica no determinística de los algoritmos genéticos, este necesita ser ejecutado un determinado número de veces para luego mediante pruebas estadísticas determinar si alguna combinación de parámetros constituye efectivamente una mejora significativa.

Debido al alto consumo de recursos computacionales de la función de fitness de cada ejecución, el ajuste de parámetros de un algoritmo genético resulta una tarea que puede demandar mucho tiempo. Sin embargo al tratarse de una tarea inherentemente paralela, se puede pensar en la utilización de computación distribuida.

5.1.3 Implementación computacional del algoritmo genético propuesto

EL algoritmo fue implementado en Python [64] por tratarse de un lenguaje de alto nivel que permite desarrollar prototipos funcionales en poco tiempo. Al existir versiones de la máquina virtual de Python para múltiples sistemas operativos, esto permite la portabilidad.

La aplicación está compuesta por una serie de archivos que conforman los distintos módulos de la aplicación.

Al tratarse de un lenguaje interpretado que se ejecuta sobre una máquina virtual los tiempos de ejecución resultan mucho mayores a los de un programa que se ejecuta de manera nativa. Para solucionar este problema se utilizó un compilador en tiempo de ejecución [65].

5.1.4 Adaptación de la aplicación al entorno Condor

Se comentan a continuación los pasos necesarios para la adaptación de la aplicación de reconocimiento de patrones de tráfico al entorno Condor. Cabe destacar que al no contar con un universo Python dentro del entorno Condor, se debe utilizar el universo Vanilla, el cual no provee muchas de las principales funcionalidades de Condor.

Para poder ejecutar el código del algoritmo genético en el entorno Condor es necesario satisfacer los siguientes requisitos:

- El código puede ser ejecutado en diferentes sistemas operativos y arquitecturas de hardware.
- El código debe poder ser fácilmente transferido al recurso remoto.

El primer requisito se satisface fácilmente, ya que en encuentran implementaciones del intérprete de Python para la mayoría de los sistemas operativos utilizados en la actualidad.

Con el objetivo de transferir los datos y el código de la aplicación, se utiliza un archivo auto descomprimible que contiene toda la información necesaria para la ejecución de la aplicación en el recurso remoto.

La aplicación es empaquetada en un único archivo usando la herramienta cx_freeze [66]. Este archivo a su vez es enlazado contra una biblioteca estática que contiene la máquina virtual de Python. Al final de este proceso se obtiene un ejecutable ubicado en un directorio previamente definido por el usuario.

El lenguaje Python cuenta con un número de módulos dinámicos que no pueden ser enlazados de manera estática al ejecutable previamente creado por cx_freeze. Por lo que, en un segundo paso, estos módulos como así también los archivos de datos requeridos por el programa, son copiados al directorio definido por el usuario. Finalmente el archivo ejecutable, los archivos de datos y los módulos dinámicos de python son incluidos en un archivo auto descomprimible utilizando la herramienta makeself.sh [67]. De esta manera se obtiene un único archivo que puede ser fácilmente transferido a un recurso remoto.

Esta herramienta permite incluir en el archivo auto descomprimible un script que contiene todos los comandos necesarios para realizar la ejecución remota de la aplicación. Se incluye además las instrucciones necesarias para la creación de los archivos de salida basados en el nombre del recurso remoto y una marca de tiempo. De esta manera se evitan posibles colisiones debido a las múltiples ejecuciones de la aplicación en un mismo nodo.

5.1.5 Resultados obtenidos

El tiempo total de ejecución de la aplicación serial en un P4 HT con 1GB de RAM es de aproximadamente 15 minutos, para validar estadísticamente cada ajuste realizado al algoritmo es necesario ejecutar el programa un mínimo de 30 veces por lo que el tiempo total de ejecución en el mismo nodo serial es cercano a las 7 horas.

En el trabajo serial [58] se presentaron los resultados obtenidos al ejecutar algoritmo durante 1200 generaciones. La elección del número de generaciones se debe principalmente a las limitaciones impuestas por los requerimientos computacionales de la función de fitness.

Los resultados obtenidos se presentan en la figura 24. Al finalizar le ejecución del algoritmo se obtuvieron reglas con atributos que permiten encontrar coincidencias sobre aproximadamente el 85% del conjunto de entrenamiento y un 80% sobre el conjunto de prueba. El histograma de la figura 25 muestra que solamente el 25% de las ejecuciones de la aplicación obtienen un porcentaje de error de clasificación cercano al 15% en el conjunto de entrenamiento e incluso se obtienen resultados cercanos al 56% de error de clasificación.

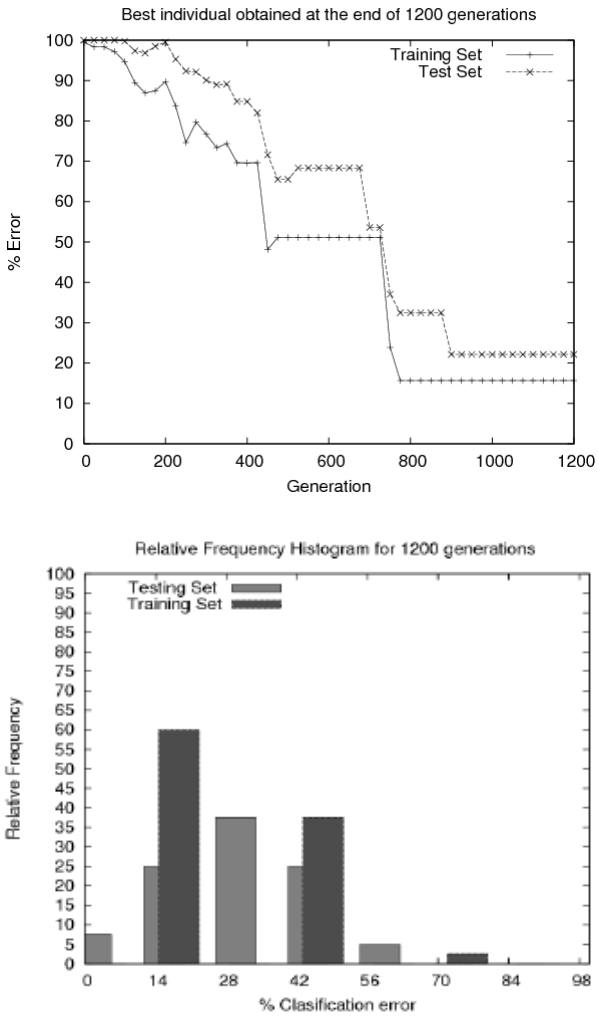

Figura 24. Resultados del mejor individuo obtenido y histograma de frecuencias relativas de la aplicación serial

La posibilidad de ejecutar la aplicación del algoritmo genético en un entorno Condor ha permitido extender el número de generaciones del algoritmo a 1.800. En este caso se utilizan todos los recursos computaciones disponibles de la figura 2, sección 2.3 del capítulo 2 de esta tesis. Cabe destacar que son necesarios tres ciclos de ejecución para ejecutar las 30 corridas del algoritmo genético propuesto, debido a que solo se cuenta con la disponibilidad de 14 nodos al mismo tiempo. En este caso el tiempo total de ejecución es de 67 minutos. Este resultado se debe a que el tiempo total de ejecución del algoritmo a lo largo de 1.800 es de 20 minutos.

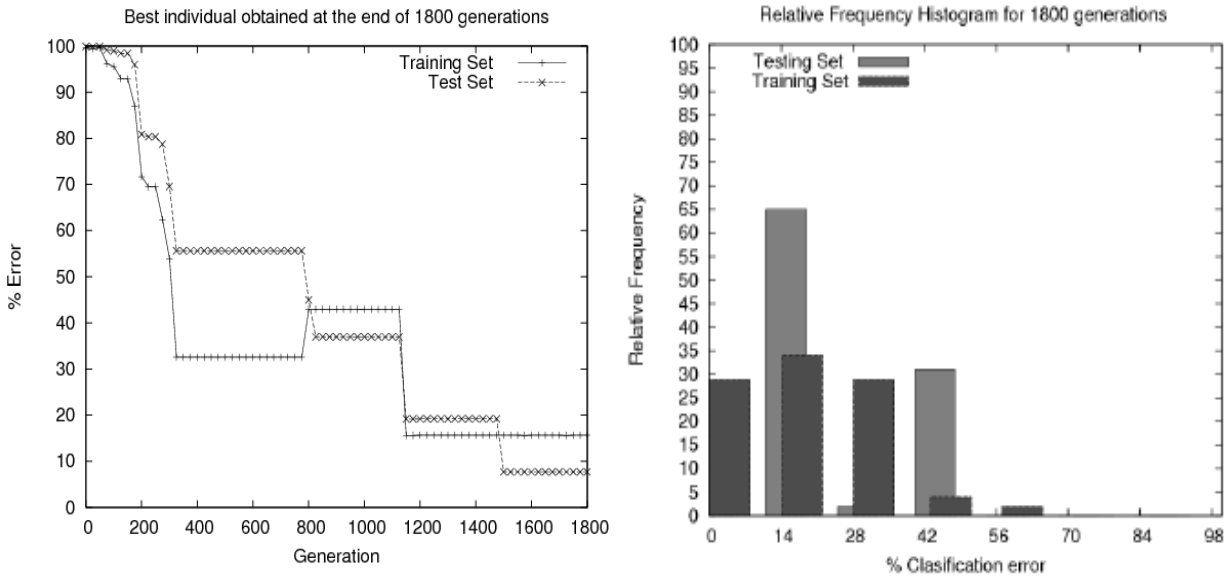

Figura 25. Resultados del mejor individuo obtenido y histograma de frecuencias relativas de la aplicación ejecutada en entorno Condor

Las pruebas preliminares del algoritmo en un entorno Condor han permitido mejorar los resultados anteriores. Luego de 1.800 generaciones se obtuvo un conjunto de reglas con atributos que permiten encontrar coincidencias sobre aproximadamente el 85% del conjunto de entrenamiento y un 93% sobre el conjunto de prueba. La frecuencia de aparición de buenas soluciones también presentó una considerable mejora ya que nunca se supera el 42% de error de clasificación en el conjunto de prueba y un 65% de las ejecuciones presentan un error de clasificación cercano al 15%.

5.1.6 Discusión de resultados

La ejecución de la aplicación bajo un entorno Condor, ha permitido obtener una reducción del tiempo total de ejecución requerido por la aplicación.

Es importante destacar que el poder computacional ofrecido por Condor ha permitido extender el número de generaciones consideradas en el algoritmo genético, obteniendo reglas de mejor calidad para el reconocimiento de patrones en el tráfico de red.

5.2 Estudios paramétricos de mecánica de sólidos en entornos de computación distribuida

Actualmente se cuenta con distintos métodos de computación distribuida capaces de incrementar los recursos de cálculos y/o disminuir tiempos de ejecución de aplicaciones intensivas, como resultan algunos casos de elementos finitos.

En este sentido se emplean cada vez más frecuentemente códigos paralelos que se procesan en distintas computadoras a partir de bibliotecas como Message Passing Interface (MPI), para lo cual se reparte el dominio de aplicación en diferentes computadoras. Sin embargo existen otras posibilidades de procesos concurrentes como es el caso de estudios paramétricos, en donde se debe ejecutar un gran número de problemas con el mismo código y diferentes datos.

Este tipo de problema, que se conoce como inherentemente paralelo, puede ejecutarse con relativa facilidad en entornos de computación de alta disponibilidad como el middleware Condor.

En esta sección se discute la ejecución paralela del código de elementos finitos SOGDE en el entorno Condor. Para instrumentar la ejecución de SOGDE sobre Condor se ha diseñado una aplicación administradora que se encarga de la orquestación del flujo de datos y tareas. La misma efectúa las tareas para generar automáticamente los ficheros de datos, procesar los mismos mediante el código SOGDE y recuperar resultados para su correspondiente edición y post-proceso, mediante herramientas adecuadas como gnuplot o similares.

De esta forma pueden realizarse estudios paramétricos mediante una ejecución única que varía los datos en forma automática y entrega un informe de resultados, listado de tablas, gráficos, etc.

El uso de la aplicación se ejemplifica mediante estudios de sensibilidad a imperfecciones de un problema de estabilidad elástica bidimensional (2D), para lo cual se escoge el caso de la columna de Euler [68]. Obtener los resultados esperados requiere procesar un gran número de problemas no lineales con SOGDE. Este trabajo fue presentado en la Asociación Argentina de Mecánica Computacional.

5.2.1 Introducción al procesamiento de problemas de mecánica computacional

En esta subsección discute el empleo del middleware en el contexto de Mecánica Computacional, con el objetivo de llevar a cabo estudios paramétricos de manera concurrente y automática.

Para procesar los problemas de Mecánica Computacional se utiliza el código SOGDE y en las referencias [69, 70, 71] se presentan los resultados más relevantes del problema aquí tratado. Un estudio paramétrico de un problema de interés requiere variar de manera adecuada los datos del problema (geometría, materiales, imperfecciones, etc.) y procesar luego diferentes problemas con cada juego de datos.

Si bien es posible realizar manualmente este tipo de estudios es una tarea sumamente tediosa, que consume gran cantidad de tiempo que el especialista puede dedicar al análisis y, además, propensa a cometer errores fruto de la envergadura de la tarea a realizar.

Condor es un middleware muy versátil para la ejecución de trabajos por lotes propia de la Computación de Alta Disponibilidad (High Throughput Computing, HTC) que en este caso permite llevar a cabo la ejecución concurrente de los diferentes procesos necesarios para obtener el estudio paramétrico.

Para complementar las capacidades de procesamiento y administración de tareas que dispone Condor es necesario diseñar una herramienta que administre y gestione la variación de parámetros, la generación de los diferentes juegos de datos, el procesamiento de los mismos y finalmente el post-proceso de los resultados obtenidos.

En el contexto Método de Elementos Finitos (MEF) resulta conocido que todo análisis basado en el método conlleva las etapas de preproceso para generar los datos, el proceso o ejecución propiamente dicha y el post-proceso de los resultados obtenidos. En este sentido Condor permite la ejecución concurrente de los diferentes procesos durante la etapa de ejecución con la consecuente disminución de los tiempos de procesamiento. Para ello es necesario contar con una infraestructura de computación distribuida adecuada como puede ser el caso de un cluster dedicado o de laboratorios de enseñanza que poseen tiempos ociosos y disponibilidad parcial.

La aplicación resultante, compuesta por la herramienta de administración, el código SOGDE y el middleware Condor, conforman una herramienta de cálculo paralelo automatizado que presenta particularidades frente a un código paralelo clásico basado en biblioteca como Message Passing Interface (MPI) por ejemplo. En este caso no es necesario modificar el código fuente que permanece exactamente igual al caso serial. La herramienta de administración propuesta complementa las capacidades de Condor y automatiza el flujo de tareas y datos necesarios para llevar a cabo el estudio paramétrico.

El código SOGDE se comenta muy brevemente en esta subsección, antes de discutir la arquitectura de la aplicación propuesta en la subsección 5.2.3. Posteriormente en la subsección 5.2.5 se evalúa la disminución de los tiempos de ejecución al utilizar a Condor como herramienta para la ejecución concurrente de procesos.

Es importante destacar que la aplicación propuesta puede emplearse en diferentes problemas que requieren el procesamiento paramétrico y que el código de cálculo, en este caso SOGDE, puede reemplazarse por cualquier otro que resulte adecuado.

El código de elementos finitos denominado SOGDE, el cual fue desarrollado por García Garino [71,72,73], emplea una formulación constitutiva basada en hiperelasticidad y en cinemática multiplicativa del tensor gradiente de deformación, pudiéndose encontrar una amplia descripción de la misma en las referencias antes mencionadas. Como caso particular posee la capacidad de encontrar trayectorias de equilibrio no lineales.

El código SOGDE ha sido ampliamente validado en la solución de problemas no lineales. En las referencias [69, 70, 71, 72] se discuten problemas que muestran comportamiento post-pico conablandamiento y endurecimiento. Además en trabajos recientes de algunos de los autores se tratan problemas de pandeo tridimensional [73,74] similares al propuesto en este caso.

Asimismo para facilitar el post-proceso gráfico de los resultados se ha incluido una interfaz con el software GID [75].

5.2.2 Ejecución de trabajos Sodge en un entorno Condor

Para procesar los trabajos de SODGE se utilizó la infraestructura presentada en la figura 4 de la sección 2.3 del capítulo 2 de esta tesis.

En la figura 4 se muestra un ejemplo sencillo de un archivo para lanzar trabajos de SODGE en Condor. En la línea 2 se indica el nombre del archivo que se desea ejecutar en Condor. Las líneas 4 y 5 permiten configurar nombres de archivos para redirigir la salida estándar y la salida de error estándar respectivamente. Mientras que en la línea 6 se indica el nombre del archivo donde Condor escribirá información relativa al estado del trabajo. Las líneas 7 y 8 indican que se debe realizar una transferencia del archivo ejecutable al nodo remoto.

En casos donde Condor ha sido instalado sobre nodos con sistemas de archivos compartidos (por ejemplo NFS) esta opción no resulta necesaria. En la línea 3 la opción Requirements permite definir los requerimientos que deben cumplir los nodos remotos para poder ejecutar el trabajo. Para el caso mostrado en el ejemplo de la figura 26 se indica que los nodos remotos deben ser nodos con arquitectura de tipo INTEL de 32 bits. Finalmente en la línea 9 se indica el número de veces que el trabajo debe ser ejecutado sobre Condor. Para este caso el trabajo va a ser ejecutando una única vez.

```
1. Universe = vanilla
2. Executable = sogde
3. Requirements = (Arch == INTEL)
4. Output = sogde.out.$(Process)
5. Error = sogde.err.$(Process)
6. Log = sogde.log.$(Process)
7. Should_transfer_files = YES
8. When_to_transfer_output = ON_EXIT
9. queue 1
```

Figura 26. Archivo que describe un trabajo de Condor

5.2.3 Arquitectura de la aplicación

Como se señaló en la subsección 5.2.1 es necesario diseñar una herramienta de administración que explote las posibilidades de Condor para resolver trabajos por lotes en forma concurrente.

El lector familiarizado con el empleo de códigos de Elementos Finitos reconocerá que en todo análisis se deben llevar a cabo tareas de preproceso, antes de la ejecución propiamente dicha del código del Método de Elementos Finitos (MEF) y el postproceso de los resultados, como se indica en la figura 27.

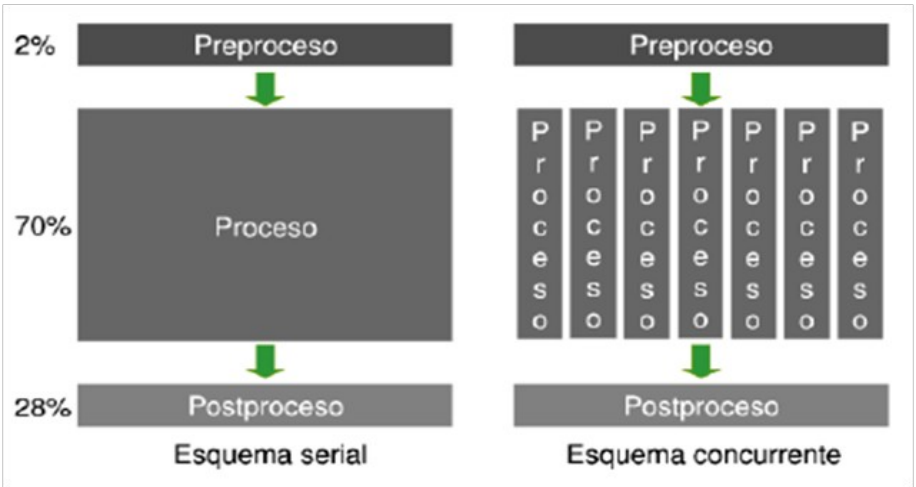

Figura 27. Esquemas serial y concurrente del proceso de un estudio paramétrico

En la figura 28 se muestra un modelo conceptual de la arquitectura de la aplicación, la cual se ha escrito en Python, ya que permite desarrollar prototipos de producción en corto plazo. En la misma se distinguen las etapas de preproceso, ejecución y post-proceso. Es importante señalar que el Master del pool se encarga de llevar a cabo todas las tareas de pre y post-proceso, mientras que la ejecución propiamente dicha se ejecuta de manera concurrente.

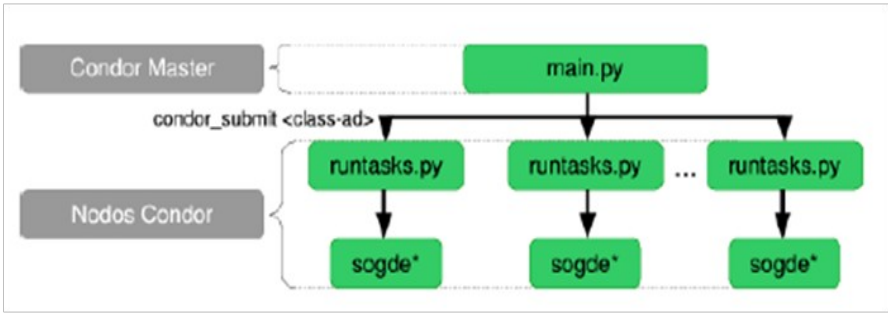

Figura 28. Arquitectura de la aplicación

El programa se inicia mediante la ejecución de un archivo que contiene la lógica principal de la aplicación, y se encarga de pasar al proceso controlador los datos de configuración necesarios para poder efectuar el barrido de parámetros.

Estos datos se encuentran en un archivo de configuración con sintaxis de Python, lo que permite realizar una importación directa de dichos datos al programa.

Una vez cargados, los datos de configuración, comienza la etapa de pre-procesamiento, en donde a partir del archivo de configuración se generan distintos archivos de datos. El proceso de ejecución de SOGDE en los nodos planificados por Condor se lleva a cabo mediante un script.

Finalmente una vez que se han procesado todos los casos de estudio, comienza la etapa de post-procesamiento en donde se procede a la extracción de datos para la generación archivos de resultados y gráficas. Se genera un archivo de log de todo el proceso, el cual contiene: i) datos del archivo de configuración especificado; ii) cantidad de casos de estudio a evaluar según cantidad de nodos y escalas y iii) evaluación de tiempos según las distintas tareas en segundos y también en porcentaje respecto del total.

A continuación se describen con mayor detalle cómo se han implementado cada una de las tres etapas citadas dentro de la aplicación de gestión.

5.2.4 Pre-procesamiento

Dado que en el estudio de problemas paramétricos se deben resolver numerosas aplicaciones con diferentes datos, en esta etapa se debe generar el conjunto de archivos para SOGDE, adecuado al problema de interés. Una aproximación sencilla sería generar en forma independiente cada archivo con un programa adecuado como GID, GMesh, o similar.

Sin embargo resulta mucho más eficiente generar dichos archivos de forma automática variando los parámetros de interés de manera automática mediante una aplicación diseñada a tal fin.

Para ello se cambian convenientemente coordenadas del modelo, con el fin de variar esbelteces y también se cambian la intensidad de la carga y su punto de aplicación con el fin de variar imperfecciones.

5.2.5 Ejecución Distribuida

La etapa de procesamiento correspondiente al modo de ejecución concurrente comienza con el armado de paquetes. Cada uno de ellos corresponde a un slice y es un archivo comprimido (tar.gz); los cuales están nombrados como sogde-package-xxxxxx.tar.gz, donde xxxxxx es un número del slice al que corresponde el paquete.]

Los paquetes contienen:

1. Un conjunto de archivos de datos (un subconjunto del total generado).
2. El solver de elementos finitos (SODGE).
3. Un archivo de parámetros para la ejecución de los procesos correspondiente para dicho paquete (slicexxxxxxData.py). Este archivo es un módulo de Pyhon cargado dinámicamente por el script runtasks.py en el nodo planificado.

La construcción de los paquetes se lleva a cabo con el fin de disminuir los tiempos de comunicación. La compresión de archivos de texto con diferencias mínimas entre ellos resulta en una disminución importante en el volumen de datos a ser transferidos por la red.

También brinda un manejo más ordenado de los archivos (que dependiendo de la configuración podrían llegar a ser muchos). Como consecuencia del empaquetado se obtiene un archivo de emisión de trabajos mucho más legible.

Con la generación de paquetes concluida se procede además al armado del archivo de emisión del trabajo de Condor, el cual se encuentra dividido en dos secciones, como se observa en la figura 29.

La sección general contiene información común de configuración para todos los trabajos a enviar. En la misma se especifica: el nombre del archivo ejecutable, que en este caso es el script runtasks.py, el universo de ejecución (Vanilla), el directorio inicial desde el cual Condor busca los archivos de entrada y en el cual

depositará todos los archivos generados por cada uno de los trabajos, el archivo de log para los trabajos y, finalmente las instrucciones para transferir los archivos de salida, lo cual se realiza al terminar cada proceso.

4. La sección de los slices contiene información particular de configuración para cada uno de los trabajos: los argumentos de ejecución para el script del trabajo (el nombre del paquete y el nombre del archivo de parámetros para el trabajo) y el archivo a ser transferido al momento de ejecutar el trabajo (el paquete).

```
# --- general section ---
Executable = runtasks.py
Universe = vanilla
Initialdir = tmp/
Log = sogde-20080630204024.log
Should_transfer_files = YES
When_to_transfer_output = ON_EXIT
# --- slices section ---
# <slice 000000>
Arguments = sogde-package-000000.tar.gz slice000000Data.py
Transfer_input_files = sogde-package-000000.tar.gz
Queue
...
# <slice 000011>
Arguments = sogde-package-000011.tar.gz slice000011Data.py
Transfer_input_files = sogde-package-000011.tar.gz
Queue
```

Figura 29. Ejemplo de ClassAd generado

Una vez generado el ClassAd se realiza la invocación del comando *condor_submit <classad>* con el cual Condor toma el control del proceso y se le delega la administración y ejecución de los trabajos. La espera de finalización de los trabajos en el programa se realiza mediante la invocación del comando *condor_wait <log-file>*, este comando no devuelve el control de programa hasta que la ejecución de todos los trabajos finalice.

La espera se hace sobre el archivo de log especificado en el ClassAd, cuyo nombre debe ser pasado como parámetro a dicho comando.

El archivo de datos adicional slicexxxxxxData.py permite organizar la ejecución de cada job manteniendo inalterado el script runtasks.py. El desacople de los datos en un archivo separado permite la existencia de un único script de ejecución independiente de los datos el cual es replicado para cada unos de los distintos trabajos a realizar.

El script comienza por la descompresión del paquete, posteriormente importa dinámicamente el archivo de datos adicionales que se encontraba dentro del paquete. Una vez que los datos fueron importados comienza la ejecución del las distintas instancias del solver indicando el archivo de datos y el archivo de salida correspondiente para cada una de los casos de estudio.

Concluido el cálculo de los distintos casos de estudio correspondientes, Condor se encarga de la transferencia de los archivos de salida, al directorio inicial especificado en el ClassAd (Initialdir = tmp/) del nodo desde el cual se enviaron los trabajos. Dichos archivos quedan entonces a disposición del script principal para proseguir con la etapa de post-procesamiento una vez que todos los trabajos sean concluidos.

5.2.6 Post-procesamiento

Con la ejecución de los cálculos de todos los casos de estudio se tienen a disposición todos los archivos de salida de cada una de las ejecuciones del solver, estos son analizados para obtener resultados de interés que permitirán generar diferentes curvas. Además se generan archivos que contienen los comandos necesarios para generar estas gráficas.

5.2.7 Estudio paramétrico de la columna de Euler 2d

Con el fin de ilustrar las posibilidades de la aplicación propuesta, se presenta la ejecución automática de un estudio paramétrico. Mediante dicho estudio se obtiene la llamada hipérbola de Euler del problema de estabilidad del equilibrio de columnas, ampliamente tratado en la bibliografía [76,77]. Para alcanzar este objetivo se deben resolver alrededor de 200 problemas no lineales con SOGDE.

En la subsección 5.2.9, define el problema y los posibles resultados a obtener con SOGDE. En la sección 5.2.10 se especifican la variación de los parámetros de interés y finalmente en la subsección 5.2.12 se discuten los resultados.

5.2.8 Definición y análisis del problema

En mecánica estructural uno de los modos de falla a considerar es la pérdida de estabilidad del equilibrio de columnas esbeltas, la cuales son sensibles a perder estabilidad debido a la aplicación de una carga de compresión en la misma.

En la figura 30 se muestra esta tipología estructural, para la situación perfecta junto a la situación imperfecta que se considera en este trabajo. Son columnas de longitud L, que se encuentran empotradas en un extremo y libre en el otro, y se le aplica una carga axial P de compresión en el extremo libre de la misma con excentricidades e respecto del eje central.

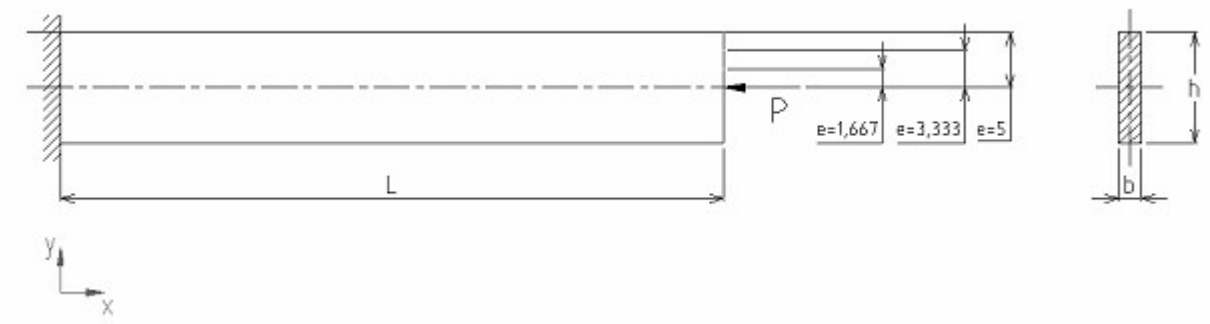

Figura 30. Geometría de la columna perfecta, vínculos y carga aplicada.

Cuando se analiza la estructura perfecta el valor de carga para el cual aparece una bifurcación del equilibrio de la trayectoria fundamental lineal, se denomina carga crítica y está dado por [77]:

$$P_{er} = \frac{\pi^2 EJ}{4L^2}$$

donde E es un parámetro del material conocido como módulo de Young y $J=bh^3/12$ el momento de inercia el cual depende de la sección de la columna. Para ese valor de carga crítica además de la trayectoria fundamental contenida en el eje longitudinal de la carga, existen configuraciones de equilibrio que tienen desplazamientos transversales no nulos. Es posible analizar el comportamiento de estructuras perfectas mediante las trayectorias de equilibrio no lineales de estructuras imperfectas asociadas. Estas últimas son obtenidas a partir de la estructura perfecta a la que se le incorpora algún tipo de imperfección, como lo son las excentricidades planteadas en la figura 30.

5.2.9 Definición del estudio paramétrico

Con el fin de estudiar la estructura para distintas longitudes L e imperfecciones e, se lleva a cabo un estudio paramétrico. Con este fin se varían las longitudes L y las excentricidades e definidas en la figura 30.

Se han analizado longitudes L que van desde L=140mm hasta L=500mm. Se ha adoptado una sección transversal uniforme de forma rectangular con una altura h=10mm y un espesor b=2mm, que se mantiene invariante en todos los casos. Con respecto a las imperfecciones, se han adoptado excentricidades con valores de e= 1,667mm, 3,333mm y 5mm.

El material empleado es un material elástico lineal con E=21000 kg/mm2 y ν=0,3. Se ha discretizado la geometría antes mencionada con 80 elementos en la longitud y 6 en la altura, empleándose en todos los casos el elemento cuadrilátero Q1 estándar [78].

El cálculo de la tensión σcr para los diferentes casos procesados permite así reproducir la hipérbola de Euler del problema.

5.2.10 Métricas

Como fue expuesto en la subsección 5.2.5, el empaquetado de archivos para su distribución en los nodos de Condor resulta ventajoso en cuanto a la disminución de los tiempos de comunicación. Con este propósito se diseña la siguiente evaluación.

Para 219 casos de estudio distintos distribuidos en 4 paquetes (aproximadamente 55 archivos de datos por paquete) tenemos que el volumen de información de cada paquete es de 968 KiB (991.546 bytes), mientras que el tamaño de todos los archivos en forma separada asciende a 2,96 MiB (3.111.215 bytes). Con lo que tenemos una disminución del volumen de datos de aproximadamente el 68,1% gracias al empaquetado. Este porcentaje varía de acuerdo a la cantidad de archivos de datos que sean incluidos en el paquete.

Otra de las ventajas del empaquetado es que se disminuye al mínimo el overhead generado por el protocolo para la preparación de la transmisión. Para realizar la transferencia de los archivos por separado dicho tiempo de preparación aparece para cada uno de los archivos a transmitir, lo cual eleva el tiempo de comunicaciones. Con el empaquetado reducimos este tiempo ya que solo se produce para cada uno de los paquetes.

En la tabla 2 se muestran para la ejecución concurrente de 219 casos de estudio en distinta cantidad de nodos, los distintos tiempos. En la primera columna el lector puede ver la cantidad de nodos utilizados, en la segunda los tiempos de duración de la etapa de proceso medida en segundos, en la tercera columna el tiempo de duración total de la ejecución del programa. En la cuarta columna se puede apreciar el porcentaje de mejora haciendo una comparación entre los tiempos de duración de la etapa de proceso en

forma serial y concurrente. En la quinta columna se puede apreciar el porcentaje de mejora de los tiempos globales de ejecución en forma serial y concurrente.

Los tiempos de demora de la ejecución serial son:

- 121 segundos en la etapa de proceso
- 177 segundos en la ejecución completa de la aplicación.

Nodos	Proceso Conc.[s]	Total[s]	Mejora Proceso	Mejora Global
2	111	166	8,26%	6,21%
3	92	146	23,97%	17,51%
4	81	135	33,06%	23,73%
5	82	137	32,23%	22,60%
6	80	135	33,88%	23,73%
7	73	128	39,67%	27,68%
8	72	126	40,50%	28,81%
9	75	129	38,02%	27,12%
10	75	130	38,02%	26,55%

Tabla 2. Métricas para ejecuciones concurrentes utilizando distinta cantidad de nodos.

De los resultados obtenidos surgen varios comentarios: el mejor tiempo de cálculo se alcanza para el caso de 8 nodos. En general el tiempo total es la suma de tiempo de proceso más tiempo de comunicaciones. A medida que aumenta el número de equipos disponibles disminuye el tiempo de proceso, pero aumenta el tiempo de comunicaciones, y en general el mínimo se alcanza en un punto intermedio, en este caso 8 equipos. Cabe plantear la posibilidad de analizar con más detalle este aspecto para tomar ventajas de un número de equipos disponibles.

Otro punto importante para discutir es el procesamiento concurrente del post-proceso, etapa que toma alrededor del 25-30% del estudio como se muestra en la figura 5. De esta manera se conseguirá disminuir aun más el tiempo de procesamiento total.

5.2.11 Discusión de resultados

En el trabajo se ha presentado una herramienta capaz de realizar complejos estudios paramétricos de manera concurrente y automatizada.

Los resultados obtenidos con el caso de estudio analizado permiten concluir que los entornos de Computación de Alta Disponibilidad (HTC) en general y Condor en particular, conjuntamente con herramientas de Mecánica Computacional como es el caso de SOGDE conducen a importantes ahorros de tiempo de procesamiento y de análisis en el caso de estudios paramétricos.

Las herramientas de HTC, como es el caso de Condor, bastante menos conocidos en nuestro país que las herramientas de HPC permiten potenciar con relativa simplicidad herramientas de Mecánica Computacional como resultan códigos de Elementos Finitos con capacidades no lineales como es el caso de SOGDE.

5.3 Experiencias en el procesamiento de aplicaciones MPI en entornos Condor

En esta sección se discute el procesamiento de aplicaciones MPI sobre recursos parcialmente dedicados, administrados por Condor (por ejemplo ciclos ociosos de trabajos de laboratorios de enseñanza). Los aspectos más importantes, así como los resultados obtenidos de este problema, se presentan en la referencia [79].

En la subsección 5.3.1 se presenta la ejecución de algunos ejemplos preliminares para conocer el rendimiento computacional de los nodos de los pool de la infraestructura dispnible. En la subsección 5.3.2 se presenta la ejecución en un entorno Condor de problemas de multiplicación de matrices y la ejecución del modelo climático MC2. Luego en la subsección 5.3.3 se muestran y discuten los resultados obtenidos.

5.3.1 Ejemplos preliminares

Para ejecutar los ejemplos mencionados en esta subsección, se utilizó la infraestructura presentada en la figura 4 de la sección 2.3 del capítulo 2 de esta tesis.

En primer lugar se ejecutó en un nodo de cada pool, un programa serial para integrar una función por el método de los trapecios [80]. Los resultados se muestran en la tabla 3. Este experimento fue desarrollado con el objetivo de calcular el índice de performance que permite medir el poder computacional de los nodos individuales del pool en relación con el front end del cluster Twister. También se calculó el índice de performance acumulado, el cual es útil para aplicaciones de escaneo de parámetros para balance de carga. Los resultados de estos experimentos se muestran en la tabla 5.

Con el objetivo de obtener la medición de otro índice de performance del pool, se ejecutó un algoritmo serial de multiplicación de matrices. Los resultados se muestran en la tabla 4. Se consideró una matriz de dimensión 4096.

Pool	Tiempo (segundos)
Twister	18
Opteron	46
Storm	61
Laboratorios	65

Tabla 3. Método de los trapecios

Pool	Características del hardware	Número de nodos	Número de núcleos	Tiempo de procesamiento	Índice de performance relativo	Índice de performance acumulado
Twister	Core 2 duo 3.0GHz 4 GB RAM	16	32	18	1	32
Opteron	Opteron 1.6 GHz 2GB RAM	4	8	46	2,55	3
Storm	Pentium 4, 3.0 GHz 1GB RAM	12	12	61	3,39	3,5
Laboratorio de Redes	Pentium 4, 2.8 GHz 1GB RAM	9	9	65	3,61	2,5
Laboratorio de Sistemas Operativos	Pentium 4, 2.8 GHz 1GB RAM	20	20	65	3,61	5

Tabla 4. Índice de performance

De acuerdo a la tabla 4, puede concluirse que el rendimiento del cluster Twister es marcadamente superior al del resto de los nodos. Todos los nodos tienen un poder computacional combinado que puede estimarse en 14 núcleos del cluster Twister. Esta diferencia de rendimiento no resulta importante si todos los nodos son utilizados para problemas de barrido de parámetros para validación estadística en Inteligencia Artificial.

Sin embargo, la realidad es muy diferente para problemas de simulación de aplicaciones gobernados por ecuaciones diferenciales, las cuales usualmente dividen el dominio espacial de interés. En este caso, la iteración “n+1” no puede ser procesada hasta que todos los cálculos de la iteración “n” hayan sido calculados. El dominio de interés de todos los nodos es dividido uniformemente entre todos los nodos (en otras palabras, asignando el mismo número de elementos finitos, nodos de diferencias finitas o equivalente en otros códigos), luego algunos nodos demorarán hasta 3.5 veces respecto del cluster Twister. Por lo tanto, en este caso, no tiene sentido agregar un nodo al cluster Twister, ya que la ejecución se verá demorada.

Pool	Tiempo (segundos)	Índice de performance relativo
Twister	1248	1
Opteron	3646	2,78
Storm	3341	2,68
Laboratorios	4678	3,75

Tabla 5. Tiempo de ejecución serial

4.3.2 Ejemplos de aplicación

Multiplicación de matrices

La multiplicación de matrices es una de las herramientas más utilizadas en el contexto de métodos numéricos en general y particularmente en problemas de simulación en ingeniería. Una de las operaciones en el cálculo de matrices es el producto de matrices. Esta operación es muy adecuada para paralelización. Los algoritmos paralelos Cartesiano [80] y Fox [81] implementan la multiplicación de matrices en entornos paralelos mediante la descomposición de la matriz en bloques o submatrices.

El algoritmo Cartesiano se caracteriza por requerir gran cantidad de memoria RAM durante su ejecución, no haciendo uso significativo de la red de comunicaciones. El algoritmo de Fox requiere pequeñas cantidades de memoria RAM, pero incrementa los tiempos de comunicación. Para obtener una descripción detallada de estos algoritmos se recomienda consultar el trabajo de Costa el al. [82] y sus referencias.

Los experimentos fueron desarrollados en los laboratorios de enseñanza, y en los pools Opteron y Storm. En el caso del cluster Twister, los experimentos fueron ejecutados en 4 nodos de un solo núcleo cada uno. Los experimentos realizados con 16 núcleos fueron desarrollados utilizand dos núcleos de los 8 nodos de Twister. La multiplicación de matrices fue ejecutada con MPI nativo y MPI sobre Condor, con el objeto de medir el overhead de Condor, que es adicionado en la ejecución de aplicaciones paralelas con MPI. La tabla 5 muestra los resultados de los experimentos computacionales de la multiplicación de matrices con los algoritmos paralelos de Fox y Cartesiano. En esta tabla, MPI significa que el trabajo ha sido ejecutado con MPI nativo, y Condor significa que ha sido ejecutado con MPI sobre Condor.

La tabla 5 muestra que la implementación del algoritmo Cartesiano para la multiplicación de matrices ofrece el menor tiempo de ejecución. El algoritmo de Fox para la multiplicación de matrices se caracteriza por hacer uso intensivo de las comunicaciones de red durante su ejecución, pero no hace uso significativo de la memoria RAM de los nodos. Los cluster Storm y Twister están conectados mediante un switch Gigabit Ethernet, por lo que en este caso donde el principal cuello de botella son las comunicaciones, estos clusters ofrecen la mejor performance, como se puede observar en la tabla 5.

Cuando estos algoritmos son ejecutados sobre 4 núcleos, se observa que incrementar la dimensión de la matriz implica incrementar el tiempo de ejecución. Cuando estos algoritmos son ejecutados en 16 núcleos, como en el caso del cluster Twister o en 16 núcleos de los pools de los laboratorios de enseñanza, se observa una disminución significativa en el tiempo de ejecución para matrices de dimensión 2048 y 4096, comparativamente con la ejecución de los algoritmos para las mismas dimensiones sobre 4 núcleos.

Los tiempos de ejecución aumentan notablemente para matrices de orden superior a los 2048, posiblemente debido a los tiempos de comunicación entre nodos.

En definitiva, si el uso de memoria no es una limitación, se prefiere el algoritmo Cartesiano ante el algoritmo de Fox, dado que el primero ofrece mejor rendimiento y menor tiempo de ejecución con respecto del último. De la tabla 5 resulta claro que el overhead que agrega Condor, respecto a la ejecución del algoritmo con MPI nativo, resulta despreciable, excepto en el caso de que el tiempo de ejecución de la aplicación en sí misma sea muy pequeño.

	4 núcleos - Laboratorio de Redes				4 núcleos - Storm			
	Cartesiano		Fox		Cartesiano		Fox	
Dimensión de la matriz	MPI	Condor	MPI	Condor	MPI	Condor	MPI	Condor
2048	35	45	137	148	25	33	112	119
4096	242	261	1130	1141	197	208	897	928
	4 núcleos – cluster Opteron				4 núcleos - Twister			
	Cartesian		Fox		Cartesian		Fox	
Dimensión de la matriz	MPI	Condor	MPI	Condor	MPI	Condor	MPI	Condor
2048	52	61	310	330	15	22	70	85
4096	368	379	2709	2760	113	133	871	908
	16 núcleos – Laboratorio				16 núcleos - Twister			
Dimensión de la matriz	MPI	Condor	MPI	Condor	MPI	Condor	MPI	Condor
2048	21	35	49	60	6	14	8	16
4096	133	169	314	350	39	50	130	139

Tabla 6. Tiempo de ejecución paralelo (segundos)

Modelo meteorológico MC2

Se ejecutó un código de investigación/producción con el objeto de experimentar sobre una infraestructura heterogénea constituida con recursos con dedicación parcial y/o total

se ejecutó el modelo meteorológico Canadian Mesoscale Compressible Community Model (MC2). En el anexo C se indican algunas consideraciones de configuración a tener en cuenta a la hora de implementar este modelo y en el anexo D se indican los pasos para su ejecución.

El modelo MC2 es un modelo completamente comprensible, no hidrostático, de área limitada capaz de auto-anidarse. Las ecuaciones del modelo MC2 están formuladas en base a las ecuaciones de Euler para el gas dentro de una esfera. Estas ecuaciones se complementan con otro conjunto de ecuaciones que incorporan el modelo físico, como ser parametrización de los cúmulos, capa límite, interacciones con la superificie terrestre, radiación y microfísica. El modelo numérico está basado en un esquema de diferencias finitas, mediante una aproximación semi-implícita y semi-Lagraniana.

El código MC2 ha sido paralelizado para mejorar su performance. Esto se logra dividiendo el dominio computacional horizontal mientras la dimensión del dominio vertical permanence inalterada. Cada subdominio computacional se distribuye entre diferentes procesadores con una región de halo (nodos fantasmas), los cuales contienen información de los subdominios límites de cada procesador vecino. Es importante mencionar que la región de halo se extiende verticalmente, ya que no se ha realizado una división vertical en el dominio computacional global. Por lo tanto se puede identificar como frentes fantasmas en lugar de nodos fantasmas.

El halo es actualizado en cada timestep mediante el intercambio de información entre procesadores vecinos, lo cual consume una cantidad muy importante de tráfico de red.

Los dominios computacionales pueden dividirse de distintas maneras (topologías) y en este estudio se han realizado distintas descomposiciones de dominios para una determinada cantidad de CPUs. Por ejemplo con 12 nodos es posible obtener distintas descomposiciones horizontales de dominios: 12x1, 2x6 y 3x4. Se experimentó con todas estas posibilidades y no se observaron mayores diferencias en el tiempo total de integración. El lector interesado puede referirse a [49] para una descripción completa de la dinámica del modelo MC2 y a [83] para una descripción completa del modelo físico MC2. En este trabajo se han analizado únicamente los resultados de 600 m para comprender las características de la tormenta.

Posteriormente al inicio de la convexión, se desarrollan dos sistemas de tormenta. Un sistema corresponde a una supercelda con movimiento hacia la derecha, mientras que la otra se mueve hacia la izquierda [84]. Estos sistemas se nombran como S1 y S2 en la figura 31. S1 se intensifica en una supercelda, mientras que S2 se propaga hacia la izquierda, como se muestra en la figura 31. Los mecanismos de formación de S1 y S2 se explican en [83].

En resumen, el desarrollo de la rotación vertical en S1 y S2 ocurre a través de la inclinación vertical del vórtice horizontal del medio ambiente, resultando en un par de vórtices. La presencia de la corriente descendiente divide el sistema de tormenta en un par rotacional ciclónico y anticiclónico. La interacción de las corrientes ascendentes con los vientos que rotan en sentido horario favorece el desarrollo de S1 y debilita a S2 (ver figura 31). En la figura 31 se observa la formación de un tornado en la punta de la tormenta S1.

El modelo MC2, por sus características, usualmente consume tiempos importantes de comunicación comparativamente con el tiempo total de ejecución. Consecuentemente, el rendimiento de la red de datos incide en los resultados que se obtengan.

El problema que aquí se presenta, ha sido procesado en clusters dedicados, pools de Condor parcialmente dedicados e instalaciones mixtas, conformadas por nodos dedicados y no dedicados. Se constituyó una infraestructura mixta agregando recursos del cluster Storm y de los laboratorios de enseñanza. En la figura 32 se muestra un gráfico log-log de los tiempos de computación requeridos para procesar el problema definido en la figura 31. Todas las curvas muestran los tiempos de ejecución de MPI sobre Condor, excepto en el caso de la infraestructura mixta de recursos dedicados y no-dedicados, donde se muestran tanto los tiempos de CPU para MPI nativo y MPI sobre Condor.

En primer lugar se muestran los resultados obtenidos utilizando recursos dedicados. En la figura 32 se observa que los tiempos de CPU se reducen a medida que el número de nodos se incrementa. Debido al número de nodos disponibles en los clusters, no se alcanzó el mínimo tiempo de CPU en los diferentes procesos ejecutados sobre los recursos dedicados. Mientras que los clusters Storm y Opteron tienen similares capacidad de CPU, el tiempo de CPU del cluster Opteron requerido para procesar la simulación del modelo meteorológico es casi 3 veces mayor que para el cluster Storm. Una explicación a este resultado puede ser que el cluster Opteron está vinculado utilizando una red Fast Ethernet y el cluster Storm está basado en una red Gigabit Ethernet.

Más aún de la comparación de los resultados obtenidos en los cluster Storm y Twister que se observan en la tabla 5, no se alcanza una diferencia importante de rendimiento. Nuevamente, en este caso los tiempos de comunicación ocultan la diferencia de rendimiento de CPU de estos clusters. El overhad que agrega Condor no resulta significativo. Este valor no se muestra en las gráficas ni en las tablas.

El mismo problema ha sido procesado utilizando una infraestructura mixta, compuesta por nodos de los laboratorios de enseñanza (recursos no dedicados) y por nodos del cluster Storm (recursos dedicados). Los resultados obtenidos se denotan como Labs_Storm_Condor para indicar ejecuciones de MPI sobre Condor y Labs_Storm_MPI para ejecuciones de MPI nativo. Los primeros 12 nodos fueron elegidos de los pools de los laboratorios para comparar con los resultados del cluster Storm.

La diferencia de tiempos de CPU a favor de Storm que se muestra en la figura 32 puede explicarse en términos de rendimiento de la red.

Es importante destacar que para los primeros 12 nodos, caso no hay una diferencia importante entre los resultados obtenidos con MPI y los resultados obtenidos con MPI sobre Condor. Los restantes nodos poseen desde 12 a 36 núcleos, los cuales son aleatoriamente elegidos por Condor, por lo que en este caso se conforma una infraestructura realmente heterogénea.

El mínimo tiempo de CPU es alcanzado aproximadamente con 16 nodos. Para una mayor cantidad de nodos, los tiempos de CPU se incrementan debido a que los tiempos de comunicación son superiores a los tiempos de procesamiento. En este caso el overhead muestra valores superiores respecto a los resultados comentados

anteriormente. Una posible razón puede encontrarse en la figura 4 de la sección 2.3, donde se muestra la infraestructura utilizada.

Es importante mencionar que deben ruterase distintas VLANs al combinar recursos de los laboratorios de enseñanza con los recursos del cluster Storm y, en este caso la latencia es superior. Por otro lado, los recursos están vinculados mediante redes de distintas velocidades, por lo que es esperable que la red más lenta degradará el tiempo total de comunicación.

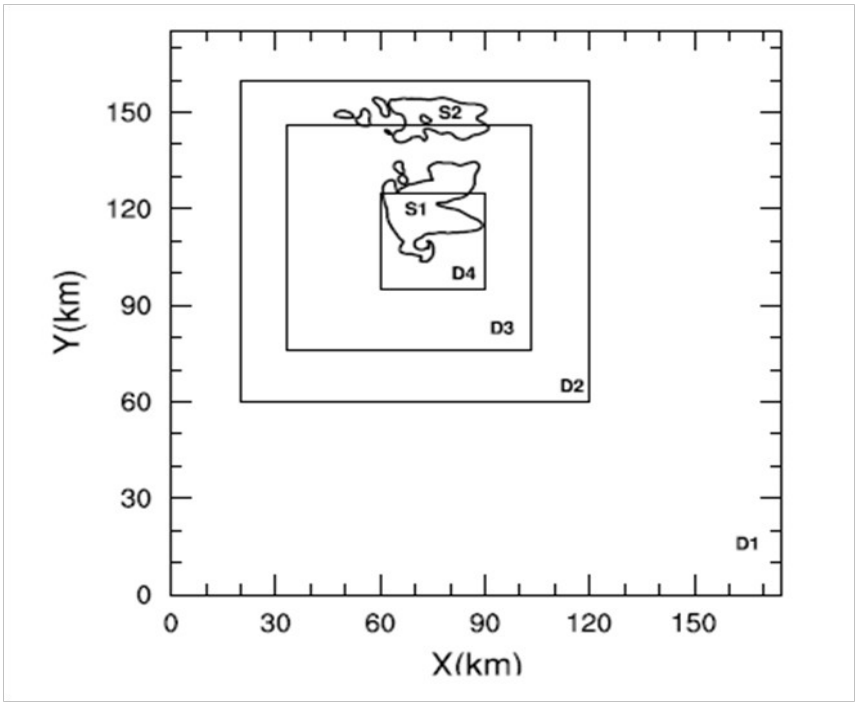

Figura 31. Los dominios de la simulación D1, D2, D3 y D4 corresponden respectivamente a los dominios de grillas con tamaños de 600 m, 200 m, 70 m y 30 m. S1 (S2) corresponde a la tormenta moviéndose hacia la derecha (izquierda)

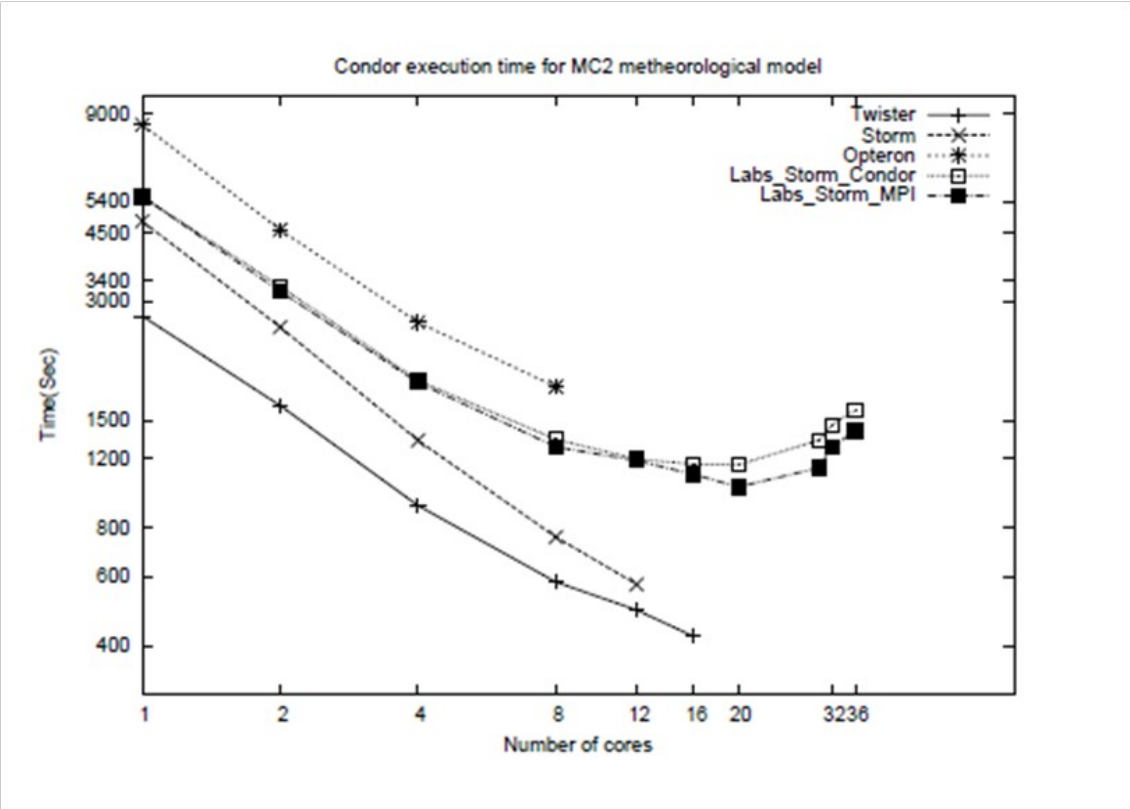

Figura 32. Tiempo de ejecución del modelo meteorológico MC2 en un entorno Condor

5.3.3 Discusión de resultados

En esta sección se ha presentado la ejecución de aplicaciones MPI en un entorno Condor. En este sentido se han llevado a cabo tareas de instalación y configuración para poder ejecutar aplicaciones MPI con Condor sobre recursos parcialmente dedicados.

Se han mencionado las ventajas de ejecutar las aplicaciones con Condor, comparativamente con su ejecución con MPI nativo.

De los resultados obtenidos de los diferentes ejemplos procesados en recursos dedicados y no dedicados y/o en una infraestructura mixta, puede extraerse que el overhead que agrega Condor no representa un incremento significativo en los tiempos de ejecución de las aplicaciones MPI paralelas.

De los experimentos procesados puede observarse que el overhead es menor y resulta acotado cuando se utilizan recursos dedicados. La misma conclusión puede extraerse cuando se utilizan recursos parcialmente dedicados. Sin embargo, para una infraestructura heterogénea el overhead resulta ligeramente superior. Una posible explicación puede encontrarse en el hecho de que se utilizan redes de diferentes

velocidades (Fast Ethernet para los laboratorios de enseñanza y Gigabit Ethernet para el cluster Storm). Por otro lado, se han ruteado diferentes VLANs, lo cual inevitablemente agrega overhead.

Las curvas correspondientes a los experimentos donde se utilizan recursos dedicados (Storm) y no dedicados (Laboratorios), muestran un corrimiento hacia arriba en el eje del tiempo. Esto es debido a que los laboratorios funcionan a 100 Mbps y están conectados a un switch a 100 Mbps, el cual se comunica con el switch principal de la institución y luego se conecta a un equipo router Linux. Desde este router se comunica el cluster Storm con una conexión Gigabit Ethernet.

Esto provoca que las comunicaciones entre el cluster Storm y los laboratorios sea a 100Mbps, aumentando los tiempos de comunicación.

A pesar de este corrimiento de la curva en el eje del tiempo, sigue resultando beneficiosa la ejecución del modelo en recursos heterogéneos, ya que se aprovecha la capacidad de cómputo de este tipo de recursos mientras los recursos dedicados están total o parcialmente ocupados.

Se observa también en las curvas correspondientes a la ejecución del modelo en recursos heterogéneos que a partir de la utilización de 20 núcleos, los tiempos de ejecución comienzan a aumentar. Esto es debido a que a partir de este punto, el tiempo de procesamiento que necesitan los núcleos es bastante inferior respecto a los tiempos de comunicación entre ellos. Esta es así debido al tamaño del modelo ejecutado.

Es importante mencionar que se ha ejecutado un código en producción denominado MC2 sobre una infraestructura administrada por Condor, sin agregar un overhead importante. Los tiempos mayores de CPU obtenidos para el código MC2 puede deberse al rendimiento de la red, para un hardware similar.

Los experimentos realizados muestran que más allá de las características bien conocidas de Condor, pueden procesarse aplicaciones MPI. De esta manera, una infraestructura de computación distribuida basada en recursos parcialmente dedicados (como los laboratorios de enseñanza o equipamiento similar), resulta una herramienta valiosa a la hora de incrementar clusters dedicados, o inclusive más importante aún, proporcionar una entrada a una infraestructura HPC.

Capítulo 6

6. Conclusiones

En la presente tesis se ha presentado al middleware Condor como un sistema para administrar los recursos disponibles en un entorno de Computación de Alta Disponibilidad. Se han realizado experimentos computacionales haciendo uso de una infraestructura de computación distribuida basada en recursos dedicados, parcialmente dedicados e infraestructuras mixtas, administradas por el middleware.

Condor es un middleware que se recomienda para usuarios que dispongan de un cluster de recursos dedicados, así como también de tiempo ocioso de CPU de estaciones de trabajo.

Se analizaron distintos gestores de trabajo, pudiéndose concluir que Condor brinda un alto poder computacional haciendo posible que cálculos con alto costo de gestión pueden ser realizados en forma ágil y eficiente, incrementando considerablemente los recursos computacionales.

Es importante mencionar la tarea de administración y configuración que implicó la conformación de los pool de Condor y la definición de los aspectos de conectividad, los cuales deben ser resueltos a la hora de configurar una infraestructura Condor. El manual del middleware disponible on line detalla los temas necesarios para su instalación y configuración, pero muchos otros aspectos relacionados sobre todos con la infraestructura de red tuvieron que ser resueltos durante la implementación.

Cabe destacar que los recursos informáticos de disponibilidad parcial de los laboratorios cuentan con distinto hardware y sistema operativo que los recursos dedicados de los cluster, lo cual implicó tareas de instalación, configuración y puesta en marcha particulares del middleware Condor. Otro aspecto a considerar es que los recursos parcialmente disponibles son utilizados gran parte del día por los alumnos de la institución, por lo que fue necesario coordinar y organizar muy bien las tareas anteriormente mencionadas y la ejecución de los experimentos computacionales.

En esta tesis se han implementado dos formas posibles de instalar Condor en un pool de nodos. La primera forma de instalación que se implementó consiste en instalar el middleware nodo por nodo y posteriormente se optimizó el proceso de instalación de Condor implementando NFS (Network File System). Se ha podido concluir que este método de instalación ofrece numerosas ventajas, por ejemplo permite acceder a archivos remotos como si fueran locales, se reducen los requerimientos de espacio de disco en las estaciones de trabajo, los archivos están centralizados, lo cual facilita su administración y ayuda a mantener la consistencia de los archivos, entre otras.

Condor permite ejecutar trabajos seriales o paralelos que requieren capacidad de cómputo intensiva. Con el fin de estudiar esta funcionalidad, se han ejecutado diversos experimentos computacionales, lográndose importantes mejoras y beneficios en todos los casos al utilizar el middleware.

Se ejecutó en un ambiente Condor una aplicación diseñada para obtener patrones de tráfico de red, para lo cual se utilizó una infraestructura con recursos dedicados. Se pudo concluir que bajo un entorno Condor, se obtiene una reducción del tiempo total de ejecución requerido por la aplicación en cuestión. Por otra parte el poder computacional ofrecido por Condor ha permitido extender el número de generaciones consideradas en el algoritmo genético de la aplicación, obteniendo reglas de mejor calidad para el reconocimiento de patrones en el tráfico de red.

Se ha ejecutado en forma concurrente el código de elementos finitos SOGDE en el entorno Condor, con el fin de realizar estudios paramétricos de mecánica de sólidos en entornos de computación distribuida. Los resultados obtenidos con el caso de estudio analizado permitieron concluir que la utilización de Condor condujo a importantes ahorros de tiempo de procesamiento y de análisis en el caso de estudios paramétricos.

Se procesaron aplicaciones MPI sobre recursos dedicados, parcialmente dedicados e infraestructuras mixtas, en entornos administrados por Condor. Se realizaron pruebas sencillas con un código de producto de matrices para demostrar el funcionamiento de Condor, luego se llevó a cabo el experimento principal de MPI, el cual consistió en simulaciones de meteorología con el programa MC2.

Resulta ventajoso ejecutar aplicaciones MPI sobre Condor en lugar de MPI nativo. En este último caso es necesario especificar en un archivo, de manera fija, los recursos sobre los cuales se ejecutará la aplicación, y de haber algún cambio deberá reflejarse en este archivo. Esta tarea puede resultar compleja cuando se cuenta con recursos con disponibilidad parcial. Al ejecutar las aplicaciones MPI sobre Condor, esta tarea es resuelta por Condor, el cual administra recursos heterogéneos y con distinta disponibilidad.

Cabe destacar que los resultados obtenidos de los diferentes ejemplos procesados demostraron que la sobrecarga que agrega Condor no representa un incremento significativo en los tiempos de ejecución de las aplicaciones MPI paralelas. Condor resulta ser una herramienta valiosa a la hora de procesar aplicaciones MPI.

Se ha detallado la forma en que Condor envía, ejecuta y monitorea los trabajos. Se ha profundizado en el mecanismo de matchmaking de Condor, mediante el cual los requerimientos de los usuarios se asocian a los recursos solicitados. Este mecanismo ofrecido por Condor se materializa mediante los avisos clasificados, los cuales son sencillos de configurar, utilizan un lenguaje flexible e intuitivo y la documentación de soporte es completa.

Se ha destacado el mecanismo de workflow que ofrece Condor, DAGMan, el cual ha permitido ejecutar aplicaciones interdependientes de manera automática.

Condor permite la implementación de variados mecanismos de seguridad, los cuales están muy bien detallados, tanto en su funcionamiento como configuración, en el manual del middleware.

En esta tesis se han ejecutado aplicaciones con códigos muy diversos entre sí. Por ello se ha limitado el acceso a los recursos, definiendo niveles de acceso, usuarios y mecanismos de autenticación y autorización. De esta manera, Condor puede asegurar que sólo usuarios de confianza accedan al pool.

También se ha detallado el modelo de Seguridad en Condor las diferentes formas de ejecutar trabajos bajo distintos usuarios con distintos privilegios, las configuraciones básicas de seguridad en Condor, descripción de los niveles de acceso y mecanismos de autenticación y autorización.

Como aporte de esta tesis, para comunicar pools de Condor separados por firewalls, se han configurado Redes Privadas Virtuales (VPN), a fin de comprobar el comportamiento del middleware a través de la VPN. Para llevar a cabo este experimento se montó una maqueta en un laboratorio de recursos informáticos con disponibilidad parcial y dos routers pertenecientes a la institución educativa.

En este diseño, los pool se encuentren en distintas redes, separados entre sí, y para Condor, a través de la VPN, esto resulta transparente, tratando a los nodos remotos como locales. Resulta importante destacar que los pool de Condor vinculados mediante la VPN fueron configurados con encriptación, certificados digitales, autenticación fuerte de usuario y control de acceso, resultando así una infraestructura altamente segura. Esto posibilita la unión de recursos computacionales de forma transparente y segura a través de Internet.

Como resultado de los experimentos y estudios desarrollados en esta tesis, se publicaron los trabajos que a continuación se detallan.

Se publicaron 3 trabajos en ENIDI (Encuentro de Investigadores y Docentes de Ingeniería), Congreso de la provincia de Mendoza, en los años 2007, 2009 y 2011.

Se publicó un trabajo en el año 2007 en el XIII Congreso Argentino de Ciencias de la Computación (CACIC), organizado en la Universidad del Nordeste, provincia de Corrientes.

En el año 2008, se presentó un trabajo en el Congreso sobre Métodos Numéricos y sus Aplicaciones, ENIEF, de La Asociación Argentina de Mecánica Computacional, AMCA.

Se publicó un trabajo en el año 2010 en 39 JAIIO, Jornadas Argentinas de Informática, 3 High Performance Computing (HPC) Symposium.

En definitiva, en esta tesis se ha comprobado que Condor es una herramienta valiosa a la hora de incrementar clusters dedicados y proporcionar una entrada a una infraestructura HPC. Condor agrega valor a recursos de disponibilidad parcial como es el caso de laboratorios de enseñanza, máquinas de escritorio y tiempo ocioso de CPU, incrementando el poder computacional de los recursos disponibles.

Anexo A. Instalación y configuración de Globus y Condor

A modo de ejemplo se configura una grid, compuesta por un nodo Condor desde el cual los usuarios emiten los trabajos a un nodo configurado con Globus, el cual a su vez emite los trabajos a nodos esclavos configurados con Condor, de acuerdo a la figura 1. Esta infraestructura está montada sobre nodos con sistema operativo CentOS.

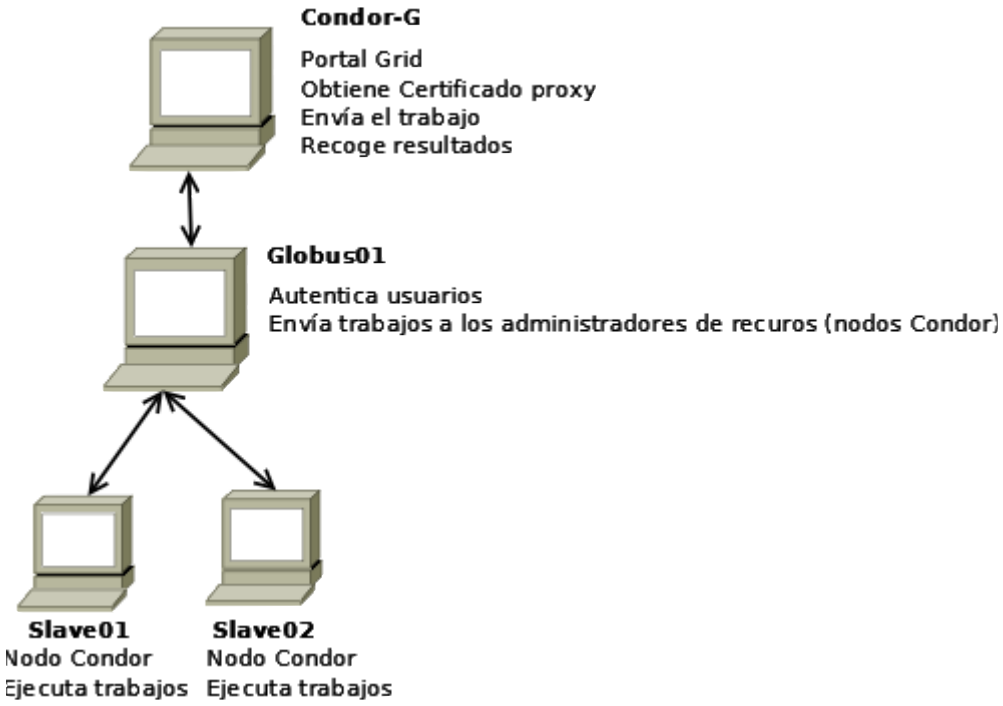

Figura 1. Arquitectura Globus-Condor

Configuración del primer nodo Globus01

a) En el primer nodo, llamado Globus01 se debe instalar una serie de paquetes previo a la instalación del Globus Toolkit, entre ellos ant, gcc, perl-XML-Parser, readline, gcc-c++, openssl-devel, ant-nodeps, xml-commons, libxml2-devel, libxml2 y xinetd. Posteriormente se instala el Globus Toolkit, que en el caso de esta infraestructura se eligió Globus Toolkit 4.2.1.

Uno de los pilares de la infraestructura grid es la seguridad. Muchos sitios web suelen mostrar un certificado firmado que aseguran al usuario que el sitio Web es de confianza. Similarmente en una grid, cada recurso debe poseer los certificados apropiados para garantizar su confianza. Estos certificados deben ser emitidos por una Autoridad Certficante (CA), la cual está encargada de emitir los certificados de los nodos y de los

usuarios de la grid. La Autoridad Certificante se puede crear utilizando el software SimpleCA de Globus y cada sistema que compone la grid debe utilizar la misma CA.

b) Creación de la Autoridad Certificante

En el nodo denominado Globus01 se crea la Autoridad Certificante, el certificado de host y la clave del contenedor.

Es necesario crear certificados para ser autenticados y autorizado, ya que las tareas que se ejecutan en los pasos posteriores así lo requieren. El Nombre Distintivo o Distinguished Name (DN) del certificado será la identidad autenticada, la cual luego será autorizada.

Se necesita identidades para los servicios y usuarios. En el caso de los servicios, se utiliza una identidad que es igual a su nombre de host. Para los usuarios, se utiliza su nombre completo. Para crear los certificados se utiliza SimpleCA, distribuido con el Toolkit de Globus.

Un contendor puede definirse como un entorno de hosting, ya que provee de un entorno de ejecución para servicios web. Administra la ejecución de servicios y recursos y sus ciclos de vida. Provee también de una infraestructura persistente de datos y seguridad, entre otras funcionalidades.

c) Selección del administrador de trabajos (Jobmanager)

Una vez que están creados los certificados, se elige el administrador de trabajos, que en este caso será Condor. Otros administradores de trabajos pueden ser PBS, SGE, LSF, etc.

d) Creación de un servidor MyProxy

En el nodo Globus01 se crea el servidor MyProxy, el cual será utilizado para almacenar los certificados. En este paso de la configuración, todavía no se han creado los certificados de usuarios, sólo los de los nodos.

Una vez que está configurado el servidor MyProxy, se puede crear los certificados de los usuarios. Para ello se deben brindar dos contraseñas, una del usuario que debe tener como mínimo 6 caracteres de longitud, y una segunda contraseña, que debe ser una contraseña simpleCA.

El usuario root deberá crear una nueva credencial en el servidor MyProxy, para lo cual deberá especificar el nombre completo del usuario y su nombre de login.

Finalmente se crea un archivo llamado grid-mapfile para autorización, como usuario root.

Grid-mapfile es un archivo que contiene entradas que asocian certificados a nombres de usuarios locales. Este archivo también se utiliza como lista de control de acceso para servicios GSI y normalmente se encuentra en `/etc/grid-security/grid-mapfile.`

El Toolkit de Globus implementa un mecanismo de autenticación mutua entre nodos para la emisión y ejecución de trabajos. La autenticación mutua requiere comparar certificados de seguridad. Los certificados de seguridad son de tres tipos:

Certificados de host

Estos certificados deben estar presentes en cada uno de los nodos que conforman la grid y son archivos de sólo lectura, inclusive para el super usuario y se almacenan en /etc/grid-security or $GLOBUS_LOCATION/etc.

Certificados de usuarios

Estos certificados se utilizan para identificar a un usuario en particular de la grid. Los certificados de usuario se almacenan por defecto en el directorio `$(HOME)/.globus.`

Certificados de Proxy

Estos certificados están totalmente disponibles y son utilizados para autenticación mutua. Estos certificados son necesarios para cada operación que se realiza con Globus. Usualmente estos certificados están ubicados en `/tmp.`

Todos estos certificados están protegidos por sus correspondientes archivos claves.

e) Configuración del servicio GridFTP

GridFTP es un protocolo de transferencia de datos ampliamente aceptado, basado en estándares, rápido, seguro y eficiente. Una vez que están creados los certificados de usuario y de host y colocados donde corresponde, se puede iniciar el servicio GridFTP en el nodo Globus01. Una vez que el servidor GridFTP está levantado, queda a la espera de peticiones, y bajo el usuario condor, se prueba la conexión al servicio.

Para utilizar GridFTP es necesario levantar el servicio xinetd. El servicio xinetd controla el acceso a un subconjunto de servicios de red populares incluyendo FTP, IMAP y Telnet. También proporciona opciones de configuración específicas al servicio para el control de acceso, registro mejorado, redireccionamiento y control de utilización de recursos.

Cuando un host cliente intenta conectarse a un servicio de red controlado por xinetd, el super servicio recibe la petición y verifica por cualquier regla de control de acceso wrappers TCP. Si se permite el acceso, xinetd verifica que la conexión sea permitida bajo sus propias reglas para ese servicio y que el servicio no esté consumiendo más de la cantidad de recursos o si está rompiendo alguna regla definida. Luego comienza una instancia del servicio solicitado y pasa el control de la conexión al mismo. Una vez establecida la conexión, xinetd no interfiere más con la comunicación entre el host cliente y el servidor.

f) Inicio del Contenedor de Servicios Web (Web Services Container)

Se inicia el Contenedor de Servicios Web en el nodo Globus01, el cual se asegura que todas las solicitudes http entrantes para un servicio web sean dirigidas a stub del servidor, el cual se encarga de interpretar las solicitudes y redirigirlas a la implementación del servicio correspondiente. Cuando la implementación del servicio obtiene un resultado, se la devolverá al stub. Tanto el stub del servidor como la implementación del servicio son administrados por el Contenedor de Servicios Web.

g) Se inicia el Servicio de Transferencia Confiable

En el nodo Globus01 se inicia el servicio RFT (Reliable File Transfer), el cual es un servicio web que brinda interfaces para controlar y monitorear transferencias de archivos de terceras partes que utilizan servidores GridFTP.

h) Configuración de GRAM

Una vez que se han configurado los aspectos de seguridad y el servidor GridFTP, se puede configurar GRAM (Grid Resource Allocation and Management) en el nodo Globus01 para administrar recursos. GRAM permite que los usuarios envíen, monitoreen y cancelen trabajos remotos en nodos de una grid. GRAM permite la ejecución de trabajos de manera remota en contextos donde es importante una operación confiable, monitoreo de las tareas y administración de credenciales.

Cabe destacar que GRAM no es un administrador de trabajos, más bien provee de un protocolo para comunicar diferentes planificadores de trabajos.

En este punto puede ejecutarse un trabajo como usuario condor, desde Globus01 con el comando globusrun-ws.

i) Configuración del servicio sshftp para acceder al servidor GridFTP mediante ssh.

El protocolo sshftp permite implementar un mecanismo de seguridad fuerte, para ejecutar una transferencia GridFTP mediante ssh.

Configuración del segundo nodo Condor-g

a) Instalación y configuración de Condor

En el nodo Condor-g debe estar instalado, configurado y probado Condor. Esto es debido a que se utilizarán las opciones --enable-wsgram-condor a la hora de ejecutar el script de configuración de Globus.

También debe estar ejecutándose Condor y los comandos de Condor deben estar disponibles para los usuarios de globus.

b) Se deben instalar todos los paquetes mencionados en el punto a) del primer nodo para luego instalar el Globus Toolkit.

c) Configuración de los parámetros de seguridad

En primer lugar se copian los certificados desde el primer nodo, Globus01, ya que el nodo Condor-g va a usar la misma CA. Luego en el nodo Globus01 se crea el certificado para Condor-g.

En el nodo Condor-g debe abrirse el puerto 7512 que corresponde al servidor MyProxy para que se comunique con el nodo Globus01.

d) Configuración de GRAM en el administrador de trabajos

Este paso se realiza una vez que los certificados de host están instalados. Al configurar GRAM se establece que el administrador de trabajos es condor.

e) Autenticación del usuario condor

Una vez creados los certificados de host para Condor-g y sabiendo que el nodo Condor-g confía en la CA (globus01), se crea un grid-mapfile para el usuario condor.

f) Configuración de GridFTP en el nodo Condor-g

La configuración de GridFTP es idéntica a la configuración en el nodo Globus01. Por lo tanto se copia desde el nodo Globus01 el archivo gridftp.

g) En este momento puede efectuarse una prueba desde el nodo Condor-g hacia el nodo Globus01, como el usuario condor:

```
# su - condor
$ myproxy-logon -s globus01
$ globus-url-copy gsiftp://condor-g/etc/group
```

gsiftp://globus01/tmp/from-condor-g

En Globus01:

 # ls -l /tmp
total 52
....
-rw-r--r-- 1 condor condor 651 Oct 6 18:22 from-condor-g
....
(success)

h) Configuración del Contenedor de Servicios Web en el nodo Condor-g

i) Configuración de GRAM en el nodo Condor-g

En este paso ya es posible enviar un trabajo Globus desde el nodo Condor-g, que será ejecutado en el nodo.

j) Emisión de trabajos Condor desde el nodo Condor-g

En la figura 2 se muestra el ClassAd de Condor para ejecutar un trabajo sencillo emitido desde el nodo Condor-g hacia en el nodo Globus01, el cual a su vez envía el trabajo a los nodos esclavos configurados con Condor. En la figura 3 se muestra el ClassAd de Condor para ejecutar un trabajo de Inteligencia Artificial para detección de patrones de tráfico en la red.

```
executable = /bin/hostname
transfer_executable = false
universe = grid
grid_resource = gt4 https://globus01/wsrf/services/ManagedJobFactoryService  Condor
error = errfile.$(Process)
output = out_test.$(Process)
machine_count = 3
globusrsl = (jobType=condor) (count=3)
log = log_test.log
should_transfer_files = no
queue 3
```

Figura 2 ClassAd de Condor

```
executable = gaids.run
transfer_executable = true
universe = grid
grid_resource = gt4
https://globus01/wsrf/services/ManagedJobFactoryService Condor
error = errfile.$(Process)
output = out_test.$(Process)
machine_count = 3
globusrsl = (jobType=condor) (count=3)
log = gaids_grid.log
should_transfer_files = yes
WhenToTransferOutput = ON_EXIT
queue 3
```

Figura 3 ClassAd de Condor

Anexo B. Instalación de MPICH2 nodo por nodo

En primer lugar se descomprime el archivo llamado mpich2-1.0.5p4.tar.bz2 en /usr/src:

tar -zxvf mpich2-1.0.5p4.tar.bz2

Se procede a la compilación e instalación del paquete:

 # ./configure
 # make

MPICH2 quedará instalado en: /usr/src/mpich2-1.0.5p4

Configuración de MPICH2 via NFS

Cabe destacar que resulta mucho más sencillo instalar MPICH2 vía NFS ya que sólo es necesario actualizarlo en un solo lugar, el maestro, resultando así fácil de mantener.

Se descomprime el archivo llamado mpich2-1.0.5p4.tar.bz2 en /opt/

Se continua con la compilación e instalación del archivo:

tar -zvxf mpich2-1.0.5p4.tar.bz2
 # ./configure
 # make

- En el directorio /etc/exports del maestro del pool se agrega la siguiente línea. Cabe destacar que existe una subred para cada cluster, Storm y Twister y otra subred

/opt/mpich2-1.0.5p4 10.10.10.0/255.255.255.0(rw,async)
/opt/mpich2-1.0.5p4 10.1.1.0/255.0.0.0(rw,async)

- Se reinicia el servicio

#service nfs restart

- En el archivo /etc/fstab de los esclavos del pool de Condor se agrega:

  storm:/opt/mpich2-1.0.5p4/ /opt/mpich2-1.0.5p4/ nfs defaults 0 0

- Finalmente se crea el directorio en los nodos esclavos y se monta el recurso compartido por NFS:

  # mkdir /opt/mpich2-1.0.5p4

  # mount -a

Anexo C. Configuraciones del modelo meteorológico Canadian Mesoscale Compressible Community Model (MC2) para ser ejecutado sobre MPI

El modelo MC2 utiliza MPI para comunicación entre los procesos. Para ello hay que crear las claves de SSH e intercambiarlas entre maestros y esclavos. Esto es un requisito para que se puedan realizar conexiones entre ellos a través de SSH sin que se solicite contraseña continuando luego la comunicación vía MPI.

- En primer lugar debe crearse el directorio /home/condor/.ssh con los permisos rwx------, con dueño y grupo condor. Para ello, como root, se realiza una conexión ssh a un esclavo, donde se creará el directorio .ssh. o bien se puede crear a mano. Los permisos de este directorio son rwx------, y el grupo y dueño es condor.

  # mkdir /home/condor/.ssh

  # chown condor.condor /home/condor/.ssh

  # chmod 700 /home/condor/.ssh

- El usuario condor debe tener clave condor, como root se escribe:

  # passwd condor

- Se crean las claves como usuario condor. Este comando se ejecuta en cada esclavo y en el maestro, una sola vez:

  $ ssh-keygen -t dsa

- Las claves quedan almacenadas en: /home/condor/.ssh/id_dsa

- Luego se procede al intercambio de claves por única vez, desde el maestro a los esclavos y entre si, todos con todos:

  # cat /home/condor/.ssh/id_dsa.pub | ssh condor@slavexx "cat >> /home/condor/.ssh/authorized_keys2"

Cabe destacar que en los esclavos los archivos authorized_keys2 y known_hosts deben tener permisos 644 y el grupo y dueño ser condor.

En el maestro los archivos id_dsa.pub y known_hosts deben poseer los permisos 644, y el archivo id_dsa permiso 600 y el grupo y dueño es condor.

Anexo D. Ejecución del modelo meteorológico Canadian Mesoscale Compressible Community Model (MC2)

A. Ejecución del modelo climático MC2 con MPI nativo

A.1 Configuración de las variables de entorno

Para poder ejecutar el modelo climático MC2 es necesario que se compartan los archivos que requiere la aplicación, a través de NFS en todos los nodos del cluster. Para este trabajo, todos los archivos se encuentran en /home/condor/nfs/mc2tst/.También debe estar presente en este directorio compartido mediante NFS, el compilador de Intel y se debe agregar la variable de entorno LD_LIBRARY_PATH=/home/condor/nfs/intel/ifc/installation/compiler70/ia32/lib/ a fin de que el modelo MC2 pueda utilizar dicha librería. La falta de este path provoca el siguiente error cuando se ejecuta el archivo ./runpilmc2.sh:

/home/condor/nfs/mc2tst/mc2ntr.Abs: error while loading shared libraries: libcxa.so.3: cannot open shared object file: No such file or directory

Cabe destacar que para ejecutar el modelo climático con MPI nativo, es necesario ejecutar el archivo ./runpilmc2.sh

Dentro del directorio donde se encuentran los archivos de la aplicación del equipo maestro debe estar presente el archivo envvars.sh. Este archivo pertenece al usuario y grupo condor y posee los siguientes permisos: -rwxr-xr-x.

Este archivo debe contener la siguiente información:

export rep_from_which_model_is_launched=`pwd`

export AFSISIO=/home/condor/nfs/mc2tst/./mc2/data/

export F_UFMTENDIAN=big

ulimit -s unlimited

export EXECDIR=`pwd`

A.2 Configuración de MPICH2

Dos archivos son necesarios para poder ejecutar el demonio mpd en cada nodo del cluster, estos son .mpd.conf y mpd.hosts. Cabe destacar que ambos archivos deben estar en todos los nodos del cluster. El archivo .mpd.conf debe pertenecer al usuario y al

grupo condor, con los siguientes permisos: -rw------- . Este archivo contiene la clave para que se ejecute el demonio mpd. El contenido del archivo es el siguiente:

MPD_SECRETWORD=condor

MPD_PORT_RANGE=500:59999

El archivo mpd.hosts contiene los hosts donde se ejecutara el trabajo y con dos puntos se indica el número de demonios mpd que se levantarán en cada host. De esta forma se pueden levantar tantos procesos mpd como núcleos tenga el nodo del cluster. El archivo mpd.hosts debe pertenecer al usuario y al grupo condor, con los siguientes permisos: -rw-r—r--.

El contenido de este archivo es el siguiente:

master:1

slave01:1

slave02:1

slave03:1

slave04:1

slave05:1

En este caso se indica el nombre del host:cantidad de demonios mpds que se ejecutaran en ese host.

A.3 Configuración y ejecución del modelo MC2

El modelo se configura desde el archivo mc2_settings.nml. En este se indica el tamaño de la grilla y la topología del cluster (número de procesadores o procesos mpd que se deben ejecutar). La topología se modifica cambiando las variables npex y npey. Por ejemplo:

#==== Topology of processors ================================

&pe_topo

npex = 2, npey = 2 #(aquí se especifica la cantidad de procesos mpd que se levantarán)

nblocx=1, nblocy=1

El archivo mc2_settings.nml debe pertenecer al usuario y al grupo condor, con los siguientes permisos: -rwxr-xr-x .

El modelo necesita crear archivos de entrada antes de ejecutarse, por esto, cada vez que se realiza una modificación en el archivo de configuración mc2_settings.nml se debe ejecutar el comando runpilmc2.sh en el nodo maestro.

Finalmente se puede ejecutar el modelo en el cluster utilizando MPI con el comando runmc2.sh.orig. Este archivo contiene:

```
#Environment variables
# Setup runtime environment variables (generated by test.sh)
. ./envvars.sh
#Environment variables end

#Run mc2
rm -rf ./tmp
mkdir $EXECDIR/tmp > /dev/null
export TMPDIR=${EXECDIR}/tmp

#Compute total no of cpus from npex,npey of mc2_settings file
cd $EXECDIR/process
ncpu=`grep npex model_settings |awk 'BEGIN{FS=","}{print $1" "$2}'|awk '{print
    expr $3*$6}'`
#Compute ends

#MPI setup (MPICH 2)
cd $EXECDIR
/usr/src/mpich2-1.0.5p4/bin/mpdboot --totalnum=4 --file=mpd.hosts --rsh="ssh"
    --user=$USER
/usr/src/mpich2-1.0.5p4/bin/mpdtrace -l
#MPI setup (MPICH 2)

#Launch MPI
/usr/src/mpich2-1.0.5p4/bin/mpirun -machinefile mpd.hosts -n $ncpu ./mc2dm.Abs
```

Cada vez que se modifica el archivo mc2_settings.nml. se debe ejecutar el archivo runpilmc2.sh el cual llama a la variable de entorno envvars.sh, borra el directorio llamado process, y luego lo vuelve a crear.

B. Ejecución del modelo climático MC2 con MPI, sobre Condor

B.1 Configuración particular de Condor para la ejecución del modelo climático MC2.

Se configura Condor, versión 7.4.1 en el maestro y en los esclavos, de acuerdo a lo indicado en el Anexo A. El archivo de configuración de Condor /opt/condor-7.4.1/etc/condor_config debe tener los siguientes parámetros:

```
TRUST_UID_DOMAIN = TRUE
UID_DOMAIN = *
FILESYSTEM_DOMAIN = $(FULL_HOSTNAME)
ALLOW_READ = *
ALLOW_WRITE = *
PARALLEL   = 11
IsParallel = (TARGET.JobUniverse == $(PARALLEL))
DAEMON_LIST= MASTER, STARTD, SCHEDD
NUM_SLOTS = 0
VM_MAX_NUMBER = 0
```

En el maestro y los esclavos, el archivo de configuración de Condor /opt/condor-7.4.1/local.master/condor_config.local debe tener los siguientes parámetros:

```
CONDOR_HOST = master
DedicatedScheduler = "DedicatedScheduler@master"
STARTD_ATTRS = $(STARTD_ATTRS), DedicatedScheduler
SLOT1_USER = condor
SLOT2_USER = condor
VM1_USER = condor
VM2_USER = condor
DEDICATED_EXECUTE_ACCOUNT_REGEXP = condor
CONDOR_IDS = 0.0
START = TRUE
SUSPEND = FALSE
PREEMPT = FALSE
CONTINUE = TRUE
WANT_SUSPEND = FALSE
WANT_VACATE = FALSE
RANK = 0
KILL = FALSE
DAEMON_LIST = COLLECTOR, NEGOTIATOR, MASTER, SCHEDD, STARTD
MPI_CONDOR_RSH_PATH = $(LIBEXEC)
CONDOR_SSHD = /usr/sbin/sshd
```

```
CONDOR_SSH_KEYGEN = /usr/bin/ssh-keygen
TRUST_UID_DOMAIN = TRUE
BIND_ALL_INTERFACES = TRUE
```

B.2 Configuración y ejecución del modelo MC2

En primer lugar, se crea el archivo para emitir el trabajo:

```
universe = parallel
executable = mp2script
log = logfile
output = outfile.$(NODE)
error = errfile.$(NODE)
Arguments = script.sh
Machine_count = 40
transfer_input_files = script.sh,  mc2dm.Abs, envvars.sh, process.tar
should_transfer_files = yes
When_to_Transfer_output = on_exit
queue
```

Cabe destacar que para poder ejecutar un trabajo MPI sobre Condor es necesario hacerlo a través de un script intermediario, llamado mp2script. Este es el encargado de realizar diversas tareas como intercambiar certificados temporales entre los nodos y el maestro a fin de que funcione correctamente mpd.

Es importante mencionar que al script original se le debieron hacer algunas modificaciones para que ejecute el modelo climático, por ejemplo se le agregó la ejecución del script de variables de entorno envvars.sh y el cálculo de la cantidad de procesos mpd a ejecutar en los nodos del cluster.

Este script es el encargado de armar el listado de nodos del cluster disponibles para ejecutar el trabajo, al cual lo arma de forma dinámica, no siendo necesario escribirlo a mano, como en el caso de la ejecución del modelo climático sobre MPI nativo.

Cada vez que un trabajo es encolado en Condor, el archivo `mpd.hosts` se arma con los nodos que en ese momento estén disponibles o que hayan cumplido con la condición de matchmaking indicada en el classad del trabajo.

A continuación se indica la configuración del archivo mp2script:

```
#!/bin/sh -x
# File: mp2script
#     Adapted from mp1script by Mark Calleja
#
#   Edit MPDIR and LD_LIBRARY_PATH to suit your local configuration.
# Also don't forget to set the secretword in .mpd.conf.
```

```
#
sh /home/condor/nfs/mc2tst/envvars.sh
export PWD=`pwd`
export MPD_CONF_FILE=~/.mpd.conf
#Cantidad de trabajos por maquina (Cant de CPUs por ejemplo)
JOBS_PER_MACHINE=1
#Dimension obtenida de las configuraciones del modelo. La saco con el path absoluto,
    tiene que estar funcionando el NFS. CUIDADO: Este numero tiene que coincidir
    con la cantidad de maquinas disponibles y con la cantidad de
    JOBS_PER_MACHINE (JOBS_PER_MACHINE x CONDOR_PROCNO)
NCPU=`grep npex /home/condor/nfs/mc2tst/process/model_settings |awk
    'BEGIN{FS=","}{print $1" "$2}'|awk '{print expr $3*$6}'`
_CONDOR_PROCNO=$_CONDOR_PROCNO
_CONDOR_NPROCS=$_CONDOR_NPROCS
CONDOR_SSH=`condor_config_val libexec`
CONDOR_SSH=$CONDOR_SSH/condor_ssh
CONDOR_SSHD="/usr/sbin/sshd"
SSHD_SH=`condor_config_val libexec`
SSHD_SH=$SSHD_SH/sshd.sh
export SCRATCH_LOC=loclocloc
echo $PWD > /tmp/$SCRATCH_LOC
CONDOR_MPI_SHELL="comando_mpi.sh"
#Este script tiene el directorio de ejecucion /opt/condor../local.../execute/dir_xxx
echo '#!/bin/sh' > $CONDOR_MPI_SHELL
echo "cd \`cat /tmp/$SCRATCH_LOC\`" >> $CONDOR_MPI_SHELL
#Aqui se pasan los argumentos del trabajo emitido con Condor,por ejemplo script.sh.
    Este script llama al ejecutable mc2dm.Abs
echo 'exec ./$@' >> $CONDOR_MPI_SHELL
chmod a+r+x $CONDOR_MPI_SHELL
rm /tmp/$CONDOR_MPI_SHELL
cp $CONDOR_MPI_SHELL /tmp
. $SSHD_SH $_CONDOR_PROCNO $_CONDOR_NPROCS
# The binary is copied but the executable flag is cleared, so the script has to take care
    of this.
EXECUTABLE=$1
shift
```

```
chmod +x $EXECUTABLE
# Set this to the directory of MPICH2 installation
MPDIR="/usr/src/mpich2-1.0.5p4/"
PATH=$MPDIR/bin:.:$PATH
export
    LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/home/condor/nfs/intel/ifc/installati
    on/compiler70/ia32/lib/"
export PATH
# Keep track where Condor's scratch dir for this VM is
finalize()
{
#  mpdallexit
 rm ~/$SCRATCH_LOC
  exit
}
trap finalize TERM
if [ $_CONDOR_PROCNO -ne 0 ]
then
    sleep 5
fi
if [ $_CONDOR_PROCNO -eq 0 ]
then
    CONDOR_CONTACT_FILE=$_CONDOR_SCRATCH_DIR/contact
    export CONDOR_CONTACT_FILE
    NODEFILE=nodefile
    myTmp=`cat $CONDOR_CONTACT_FILE | cut -f 2,3 -d ' '| tr -s [:blank:] ":" |
    sort -u`
    rootHost=`hostname`
    touch $NODEFILE
for i in $myTmp;
    do
            nombre=`echo $i | cut -f 1 -d ':' | tr -d [:space:]`
            #port=`echo $i | cut -f 2 -d ':' | tr -d [:space:]`
            echo $nombre:$JOBS_PER_MACHINE >> $NODEFILE
    done
```

```
    nodes=`wc -l $NODEFILE | cut -f 1 -d ' '`
    # The second field in the contact file is the machine name that condor_ssh knows
    how to use. Note that this used to say "sort -n +0 ...", but -n option is now
    deprecated.
    sort < $CONDOR_CONTACT_FILE | awk '{print $2}' > machines
    $CONDOR_CHIRP put $_CONDOR_SCRATCH_DIR/contact
    $_CONDOR_REMOTE_SPOOL_DIR/contact
    mpdboot --totalnum=$nodes -v -f $NODEFILE
    val=$?
    if [ $val -ne 0 ]
    then
echo "mpdboot error : $val"
            exit 1
    fi
    mpdtrace -l
    echo $CONDOR_MPI_SHELL $EXECUTABLE $@
    mpiexec -machinefile $NODEFILE -n $NCPU /tmp/$CONDOR_MPI_SHELL
    $EXECUTABLE $@
fi
```

7. Referencias

[1] Thain D, Tannenbaum T, y Livny M, Distributed Computing in Practice: The Condor Experience. Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, páginas 323-356, Febrero y Abril, 2005 http://research.cs.wisc.edu/htcondor/publications.html

[2] Livny M., Basney J., Raman R. y Tannenbaum T. Mechanisms for High Throughput Computing, Computer Sciences Department, University of Wisconsin-Madison, http://research.cs.wisc.edu/htcondor/publications.html.

[3] Condor World Map. Disponible en http://research.cs.wisc.edu/htcondor/map/map.world.color.large.gif.

[4] Running Stochastic Models on HTCondor, disponible en http://www.hpcinthecloud.com/hpccloud/2013-05-30/running_stochastic_models_on_htcondor.html

[5] Legion. Disponible en http://legion.virginia.edu

[6] Platform, LSF Load Sharing Facility. Disponible en http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/

[7] PBS. Disponible en http://www.pbsworks.com/?AspxAutoDetectCookieSupport=1

[8] OpenPBS. Disponible en http://www.pbsworks.com/

[9] Torque. Disponible en http://www.adaptivecomputing.com/products/open-source/torque/

[10] Oracle Grid Engine. Disponible en http://www.oracle.com/technetwork/oem/host-server-mgmt/ds-gridengine-167114.pdf

[11] UNICORE. Disponible en http://www.unicore.eu/

[12] SLURM: Disponible en: https://computing.llnl.gov/linux/slurm/

[13] P. Martínez, C. García Garino, C. Catania & J. Monetti: Experiencias en computación de alta disponibilidad con el entorno Condor. Desarrollos e Investigaciones Científico-Tecnológicos en Ingeniería, Maldonado, G., Veca, A. y Cremades, H. (Eds.), Anales del Congreso ENIDI 2007, pp. 157-163, Fac. Regional Mendoza, Universidad Tecnológica Nacional, 2007, ISBN 978-950-42-0087-1.

[14] P. Martínez, C. Catania, C. García Garino y J. Díaz: Reconocimiento de patrones de tráfico de red en un ambiente Condor. 1288-1299, VIII Workshop de Procesamiento Distribuido y Paralelo. Anales del CACIC 2007, XIII Congreso Argentino de Ciencias de la Computación, Corrientes, Universidad del Nordeste, 2007, ISBN 978-950-656-109-3.

[15] Instalación de Condor. Disponible en: http://research.cs.wisc.edu/htcondor/manual/v8.0/3_2Installation.html

[16] Configuración de Condor. Disponible en:

http://research.cs.wisc.edu/htcondor/manual/v8.0/3_3Configuration.html

[17] Ali R. Butt, R. Zhang, Y. Charlie Hu. A self organized pool of Condor. School of Electrical and Computer Engineering, Purdue University, 2005

[18] Condor-G, the gt2, gt4, and gt5 Grid Types. Disponible en http://research.cs.wisc.edu/htcondor/manual/v8.0/5_3Grid_Universe.html

[19] The Globus Alliance. Disponible en http://www.globus.org/

[20] Glidein. Disponible en

http://research.cs.wisc.edu/htcondor/manual/v7.6/3_3Configuration.html

[21] Condor's Security Model. Disponible en http://research.cs.wisc.edu/htcondor/manual/v8.0/3_6Security.html#SECTION00461000000000000000

[22] Niveles de acceso en Condor. Disponible en http://research.cs.wisc.edu/htcondor/manual/v8.0/3_6Security.html#SECTION00461100000000000000

[23] Kerberos: The Network Authentication Protocol. Disponible en http://web.mit.edu/Kerberos/

[24] GSI, Grid security Infraestructure. Disponible en http://www.globus.org/security/overview.html

[25] SSL, Secure Socket Layer. Disponible en http://www.verisign.com/ssl/ssl-information-center/how-ssl-security-works/

[26] Autenticación. Disponible en

http://research.cs.wisc.edu/htcondor/manual/v8.0/3_6Security.html#SECTION00463000000000000000

[27] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Disponible en http://www.ietf.org/rfc/rfc3280.txt

[28] Condor Connection Brokering (CCB). Disponible en

http://research.cs.wisc.edu/htcondor/manual/v8.0/3_7Networking_includes.html#SECTION00474000000000000000

[29] Generic Connection Brokering (GCB). Disponible en

http://research.cs.wisc.edu/htcondor/doc/CondorandFirewalls.pdf

[30] OpenVPN. Disponible en http://openvpn.net/

[31] M. Hall. Performance Analysis of OpenVPN on a Consumer Grade Router. Disponible en http://www.cse.wustl.edu/~jain/cse567-08/ftp/ovpn/index.html.

[32] Blowfish. Disponible en http://en.wikipedia.org/wiki/Blowfish_%28cipher%29

[33] D. Elminaam, H. Abdual Kader, y M. Hadhoud. Evaluating The Performance of Symmetric Encryption Algorithms. Higher Technological Institute 10th of Ramadan City, Egypt, Faculty of Computers and Information Minufiya University, Egypt, 2009.

[34] Livny, M. Raman, R. Foster, I. y Kesselman, C. (ed.) High-throughput resource management. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1998.

[35] Classified Advertisements. Disponible en

http://research.cs.wisc.edu/htcondor/classad/classad.html

[36] Raman, R. Livny, M. y Solomon, M. Matchmaking: Distributed Resource Management for High Throughput Computing, Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing. 1998, Chicago, Illinois.

[37] Estado de las máquinas, disponible en

http://research.cs.wisc.edu/htcondor/manual/v8.0/3_5Policy_Configuration.html#SECTION00456000000000000000

[38] Universos de Condor. Disponible en

http://research.cs.wisc.edu/htcondor/manual/v8.0/2_4Road_map_Running.html

[39] Protocolo Chirp. Disponible en:

http://research.cs.wisc.edu/htcondor/manual/v8.0/2_8Java_Applications.html#SECTION00383000000000000000

[40] Java Applications. Disponible en:

http://research.cs.wisc.edu/htcondor/manual/v8.0/2_8Java_Applications.html

[41] MPI Forum, http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf

[42] Wright, D, Cheap cycles from the desktop to the dedicated cluster: combining opportunistic and dedicated scheduling with Condor,Department of Computer Sciences, University of Wisconsin, Madison, 2001.

[43] Mp2script. Disponible en
http://www.ucs.cam.ac.uk/scientific/camgrid/documents/mpich2-smp/view

[44] Rocks. Disponible en http://www.rocksclusters.org/wordpress/

[45] OpenMPI. Disponible en http://www.open-mpi.org/

[46] MPICH. Disponible en http://www.mcs.anl.gov/research/projects/mpich2/

[47] Parallel Applications (Including MPI Applications). Disponible en
http://research.cs.wisc.edu/htcondor/manual/v8.0/2_9Parallel_Applications.html

[48] Mpiexec command, Ohio Supercomputer Center. Disponible en
https://www.osc.edu/~djohnson/mpiexec/

[49] Benoit R., Desgagné M., Pellerin P., Pellerin S., Chartier Y. and Desjardins S.: The Canadian MC2: A semi-implicit semi-Lagrangian wide-band atmospheric model suited for fine-scale process studies and simulation. MWR, vol. 125, pp 2382-2415, 1997.

[50] Submitting Jobs Without a Shared File System: HTCondor's File Transfer Mechanism. Disponible en
http://research.cs.wisc.edu/htcondor/manual/v8.0/2_5Submitting_Job.html

[51] Managing a Job. Disponible en
http://research.cs.wisc.edu/htcondor/manual/v8.0/2_6Managing_Job.html

[52] Priorities and Preemption. Disponible en
http://research.cs.wisc.edu/htcondor/manual/v8.0/2_7Priorities_Preemption.html

[53] Standalone Checkpointing, disponible en http://www.cs.wisc.edu/condor/manual/v7.5/4_2Condor_s_Checkpoint.html#SECTION00521000000000000000

[54] DAGMan Applications. Disponible en http://research.cs.wisc.edu/htcondor/manual/v7.8/2_10DAGMan_Applications.html

[55] P. Couvares, T. Kosar, A. Roy, J. Webber, K. Wegner. Workflow Management in Condor. University of Wisconsin-Madison, Computer Sciences Department and Louisiana State University, Department of Computer Science and Center for Computation & Technology, 2007. Disponible en http://www.springerlink.com/content/r6un6312103m47t5/

[56] Red Hat Documentation. Chapter 15 DAGMan, disponible en https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_MRG/2/html/Grid_User_Guide/

[57] P. Martínez Paula, C. Catania, C. García Garino y J. Díaz: Reconocimiento de patrones de tráfico de red en un ambiente Condor, 1288-1299, VIII Workshop de Procesamiento Distribuido y Paralelo. Anales del CACIC 2007, XIII Congreso Argentino de Ciencias de la Computación, Corrientes, Universidad del Nordeste, 2007, ISBN 978-950-656-109-3.

[58] Catania C. y Garcia Garino C. Una propuesta de reconocimiento de patrones en el tráfico de red basada en algoritmos genéticos the 9th Argentinian Symposium on Artificial Intelligence, ASAI, 2007.

[59] Bridges S. and Vaughn R. Fuzzy data mining and genetic algorithms applied to intrusion detection. In National Information Systems Security Conference, 2000.

[60] Gong R., Zulkernine M., and Abolmaesmumi P. A software implementation of a genetic algorithm based approach to network intrusion detection. In Sixth SNPD/SAWN, 2005.

[61] Li W. and Traore I. Detecting new forms of network intrusion using genetic programming. Computational Intelligence, Vol 20, 475494,

2004.

[62] Mukherjee B., Heberline L. T. and Levitt K. Network instruction detection. IEEE Network, 1994.

[63] Mitlincoln laboratory DARPA dataset. Disponible en:

http://www.ll.mit.edu/IST/ideval/data/dataindex.html.

[64] Python Programming Language. Disponible en http://www.python.org.

[65] Rigo R. Psyco, the python specializing compiler. Disponible en:

http://psyco.sourceforge.net/psyco.ps.gz, 2001.

[66] C. C. Ltd. cx freeze, a set of utilities for freezing python scripts into executables. Disponible en: http://python.net/crew/atuining/cx_Freeze/.

[67] Peter S. makeself, make self-extractable archives on unix. Disponible en:

http://www.megastep.org/makeself/.

[68] C. A. Catania, C. Careglio, D. Monge, P. Martínez, A. Mirasso y C. García Garino: Estudios Paramétricos de Mecánica de Sólidos en Entornos de Computación Distribuida, 1063-1084, Mecánica Computacional, Vol. 27, A. Cardona et al. (compiladores), AMCA, Santa Fe. Argentina, 2008, ISSN 1666-6070.

[69] García Garino, C., Un modelo numérico para el análisis de sólidos elastoplásticos sometidos a grandes deformaciones. PhD. Thesis, E.T.S. Ingenieros de Caminos, Universidad Politécnica de Catalunya, 1993.

[70] García Garino, C., y Oliver, J., Un modelo constitutivo para el análisis de sólidos elastoplásticos sometidos a grandes deformaciones: Parte i formulación teórica y aplicación a metales. Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería, 11-No.1: 105-122, 1995.

[71] García Garino, C., y Oliver, J., Un modelo constitutivo para el análisis de sólidos elastoplásticos sometidos a grandes deformaciones: Parte ii implementación numérica y ejemplos de aplicación. Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería, 12-No.2: 147-169, 1996.

[72] García Garino, C., Gabaldón, F., y Goicolea, J. M., Finite element simulation of the simple tension test in metals. Finite Elements in Analysis and Design, 42:1187–1197, 2006.

[73] Careglio, C., Mirasso, A., y García Garino, C., Estabilidad del equilibrio elastoplástica con elementos finitos y cinemática de grandes deformaciones. Mecánica Computacional, XXV: 1947-1960, ISSN 1666-6070. AMCA, 2006.

[74] Careglio, C., Mirasso, A., y García Garino, C., Estudio numérico de una columna cruciforme en grandes deformaciones. Mecánica Computacional, XXVI: 129-143, ISSN 1666-6070. AMCA, 2007.

[75] Ribó, R., y et. al. GID User Manual, Versión 8. CIMNE, 2006.

[76] Bleich, F., Buckling Strength of Metal Structures. McGraw Hill, 1952.

[77] Timoshenko, S., Resistencia de materiales. Espasa Calpe S.A., 1982.

[78] Zienkiewicz, O. C. y Taylor, R. L., The finite element method, volumes I y II. McGraw Hill, 1991.

[79] Paula Martínez, Jorge Rubén Santos, Emmanuel Millán Kujtiuk, Carlos Catania, Javier Díaz y Carlos García Garino, “Experiences in processing MPI application in Condor enviroments”, 39 JAIIO, Jornadas Argentinas de Informática, 3 HighPerformance Computing (HPC) Symposium. Editores: Ricardo Orosco (UADE), Alejandro Fernandez (LIFIA, Facultad de Informática, UNLP) Buenos Aires, 2010. Anales de las 39JAIIO - ISSN 1850-2776.

[80] Pacheco P. S.:Parallel Programming with MPI. Morgan Kaufmann D., Publishers, Inc. 1997.

[81] Grama A., Karypis, G., Kumar V. and Gupta A., et.al.: Introduction to Parallel Computing. Addison-Wesley, 2003.

[82] Costa N., Catania C., García Garino C.,León O. and Silva M.: Cálculo de Productos Matriciales en Cluster Beowulf. II Encuentro de Investigadores y Docentes de Ingeniería. Desarrollos e Investigaciones Científico-Tecnológicos en Ingeniería. Rivera, S. and Núñez Mc Leod J. (eds.) Facultad de Ingeniería, Universidad Nacional de Cuyo, pp. 37-44, 2006.

[83] Mailhot J., Belair S., Benoit R., Bilodeau B., Delage Y., Fillion L., Garand L., Girard C., and Tremblay A.: Scientific description of the RPN physics library - Version 3.6, Recherche en Prevision Numerique, Atmospheric Environment Service, Dorval, Quebec, pp. 188, 1998.

[84] Klemp J. B. y Rotunno R.: Dynamics of tornadic thunderstorms. ARFM, vol. 19, pp 369-402, 1987.