Reconocimiento de patrones de tráfico de red en un ambiente Condor

Paula Martínez, Carlos Catania y Carlos García Garino
LAPIC, Instituto Tecnológico Universitario, Universidad Nacional de Cuyo
Mendoza, 5500, Argentina
pmart@uncu.edu.ar, {ccatania,cgarcia@itu.uncu.edu.ar}

y

Javier Díaz
LINTI, Facultad de Informática, Universidad Nacional de la Plata
Calle 115 entre 49 y 50, La Plata, Argentina
jdiaz@unlp.edu.ar

Resumen

La clave de la computación de alta disponibilidad es hacer un uso eficiente de los recursos disponibles. En este trabajo se describen las características y funcionalidades del software Condor que permite la administración y explotación eficiente de los recursos en dichos entornos.
Se discute la ejecución en un ambiente Condor de una aplicación diseñada para obtener patrones de tráfico de red. Además, presenta la infraestructura del sistema sobre la que se ejecuta dicha aplicación concurrente. 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, las mejoras obtenidas muestran la capacidad de Condor como un sistema por lotes de computación distribuida de alta disponibilidad.

Palabras clave: Condor – HTC – IDS – algoritmos genéticos – tráfico de red

1. INTRODUCCIÓN

En la actualidad la potencia de cálculo es un factor a tener en cuenta en la realización de todo proyecto de investigación que requiere recursos computacionales. En muchos casos se necesita 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 computación distribuida es posible atacar problemas difíciles de resolver un tiempo atrás (debido al tiempo necesario o al costo de la infraestructura de computación a emplear). Se pueden distinguir dos paradigmas de empleo de un cluster:

- 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 son parte de este ámbito, y su aplicación directa pasa por la realización de la mayor cantidad de trabajos a lo largo del tiempo.

El paradigma de 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 [1].
Condor [2] es un entorno distribuido 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 de Condor es hacer posible que cálculos con alto costo de gestión puedan ser realizados de forma ágil y eficiente.
En este trabajo se discuten los cambios necesarios para la adaptación al entorno Condor de una aplicación de reconocimiento de patrones en el tráfico de red basada en algoritmos genéticos [3]. 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.
El trabajo se organiza de la siguiente manera: en la segunda sección de este se describe la infraestructura para ejecutar trabajos con Condor. Posteriormente, en la sección tres se detallan las características más interesantes de Condor y su funcionamiento general.
La gestión y administración de Condor, que incluye tareas como el acceso de los usuarios al conjunto de máquinas administradas por Condor, los mecanismos de autenticación y autorización de usuarios, las políticas de ejecución de trabajos, el mecanismo de checkpointing, etc., se discuten en la sección cuatro.
En la sección cinco se describen los archivos de emisión de los trabajos, los entornos de ejecución (universos) que soporta Condor y se ejemplifica la ejecución de un trabajo en Condor sobre la infraestructura disponible. La aplicación para reconocimiento de patrones en el tráfico de red se discute en la sección seis y finalmente se brindan las conclusiones en la sección siete.

2. INFRAESTRUCTURA

En esta sección se describe la infraestructura disponible para ejecutar trabajos de Condor El mismo incluye el cluster Reloaded y la computadora llamada Glapic (ver Figura 1) que se comporta como front end.
Glapic posee un P4 HT 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.
El cluster Reloaded tiene una estación de trabajo maestra que actúa como front end y 12 estaciones de trabajo esclavas. Estas estaciones conforman lo que se conoce como pool en la nomenclatura de Condor.
Las estaciones poseen un procesador P4 HT 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.

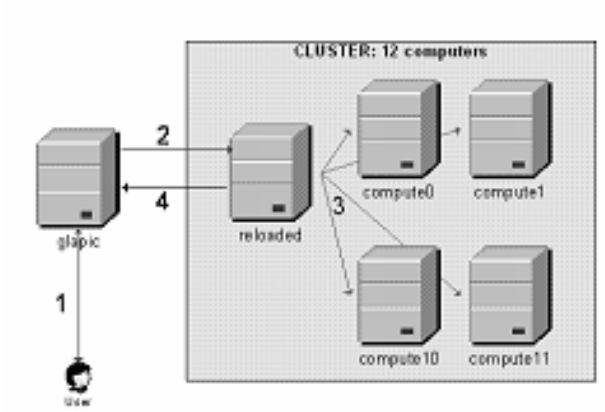

Figura 1. Cluster Reloaded

Glapic cuenta con Fedora Core 3 Kernel 2.6.9 como sistema operativo y con Condor 6.8.4 como middleware. El cluster Reloaded cuenta con la distribución para cluster Rocks 4.1.

3. CARACTERÍSTICAS Y FUNCIONAMIENTO DE CONDOR

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

- Sistema de colas integrado. El trabajo se envía a Condor. El entorno escoge el nodo más adecuado 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 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 asociar los ClassAd de trabajos con los de recursos.
Una vez que Condor decide ejecutar un trabajo, transfiere tanto el programa como los ficheros necesarios para su ejecución al recurso adecuado 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 permite el checkpointing, que consiste en guardar de forma periódica el estado completo de una tarea para poder recuperarla a posteriori.

4. ADMINISTRACIÓN DE RECURSOS

En esta sección se detalla la forma en que Condor administra la ejecución de trabajos, como resuelve el encolamiento, planificación y checkpointing de los trabajos, la asociación de solicitudes de recursos con las ofertas de recursos y aspectos de seguridad.
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, este los encola, posteriormente los ejecuta e informa los resultados al usuario [2].

4.1 Acceso al pool

En la arquitectura disponible para ejecutar trabajos bajo Condor (ver Figura 1), el administrador central que recoge toda la información del pool es Glapic y tiene derechos de administrativos sobre el mismo actuando como negociador entre los recursos y las solicitudes de recursos. Eventualmente se podría acceder directamente a Reloaded, pero el esquema actual permitirá manejar desde Glapic otros recursos además del cluster.
Estas tareas las ejecutan distintos demonios. Los demonios de Condor se ejecutados bajo el usuario root.
Por otra parte los usuarios se conectan al administrador central con el usuario condor, y desde esta máquina central se emiten los trabajos al pool.
Esta máquina cumple un rol muy importante en el pool, debe ser confiable y está en funcionamiento permanente. Si este equipo deja de trabajar, no podrán realizarse asociaciones entre recursos y requerimientos de recursos.

4.2 Autenticación y autorización de usuarios

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. Para el usuario condor que se ha creado, 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, en el archivo de configuración general de Condor llamado `condor_config`, 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.
Los administradores centrales realizan la asociación de los recursos con trabajos y recogen información de todo el pool. Las máquinas del pool anuncian sus características en avisos clasificados (ClassAds), así como también los trabajos anuncian sus requerimientos y preferencias. Un ejemplo de esto es presentado en la Tabla 1

Tabla 1. Ejemplo de ClassAdd

```
NameOpSysArch State Activity LoadAv Mem ActvtyTime
vm1@compute-0 LINUX INTELOwner Idle 1.000 504 0+00:55:08
vm2@compute-0LINUX INTEL Owner Idle 2.000 504 0+00:19:34
vm1@compute-0LINUXINTELUnclaimedIdle0.0005040+00:15:04
vm2@compute-0LINUXINTELUnclaimedIdle0.0005040+00:18:18
vm1@compute-0LINUX INTELOwner Idle1.000 504 0+00:50:07
vm2@compute-0LINUX INTEL Owner Idle 2.000 504 0+00:15:11
vm1@compute-0LINUX INTEL Owner Idle 1.000 504 0+00:50:13
vm2@compute-0LINUX INTEL Owner Idle 2.060 504 0+00:20:05
vm1@compute-0LINUX INTEL Owner Idle 1.000 504 0+00:45:34
vm2@compute-0LINUX INTEL Owner Idle 2.000 504 0+00:35:31
vm1@compute-0LINUX INTEL Owner Idle 1.000 504 0+00:45:10
vm2@compute-0LINUX INTEL Owner Idle 2.000 504 0+00:20:39
vm1@reloadedLINUXINTELUnclaimedIdle 0.010 504 0+00:17:30
vm2@reloadedLINUXINTELUnclaimedIdle 0.000 504 0+00:17:29
```

El demonio `condor_negotiator` es el responsable de las asociaciones entre recursos y solicitudes de recursos dentro del pool. Este demonio inicia periódicamente un ciclo de negociación, consultando a la máquina que actúa como recolector de información del pool (es decir el collector), sobre el estado de los recursos.
El demonio `condor_schedd` representa las solicitudes de recursos dentro del pool. Los demonios `condor_collector` y `condor_negotiator` se ejecutan en el administrador central.
El demonio `condor_collector` es el responsable de recoger información sobre el estado del pool.

4.3 Ejecución de trabajos

Para procesar un programa en Condor, se debe ejecutar un trabajo por lotes, en background. Con este fin es necesario crear los archivos que contienen las entradas correspondientes para el programa a ejecutar.
En el archivo de emisión del trabajo (ver Figura 2), se especifica si Condor debe transferir los archivos necesarios para la ejecución del trabajo.
En la Figura 2, `Requirements` indica requerimientos específicos y `Rank` indica preferencias, para que dentro de los recursos que cumplan los requerimientos, el trabajo se ejecute en máquinas determinadas.
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 la máquina que emite el trabajo a la máquina remota donde se ejecuta. y `when_to_transfer_output` que especifica cuándo transferir los resultados [4].

Así, Condor transfiere los archivos al nodo, lo ejecuta y transfiere la salida a la máquina que emitió el trabajo.

```
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 2. Archivo de emisión de un trabajo

Se puede saber dónde se está ejecutando el trabajo con el comando `condor_q`, de acuerdo a lo especificado en la condición `Rank` del archivo de emisión del trabajo. Un ejemplo de la salida de `condor_q` se muestra en la Tabla 2

Tabla 2. Salida del comando condor_q

```
ID OWNER SUBMITTED RUN\_TIME HOST(S)
28.0 condor 6/8 10:49 0+00:27:54 vm2@-0-1.local
28.1 condor 6/8 10:49 0+00:27:57 vm2@0-8.local
28.2 condor 6/8 10:49 0+00:33:05 vm1@reloaded
28.3 condor 6/8 10:49 0+00:33:02 vm2@reloaded
28.4 condor 6/8 10:49 0+00:27:54 vm2@0-5.local
28.5 condor 6/8 10:49 0+00:18:08 vm2@0-3.local
28.6 condor 6/8 10:49 0+00:18:06 vm2@0-2.local
28.7 condor 6/8 10:49 0+00:07:42 vm2@0-9.local
28.8 condor 6/8 10:49 0+00:06:17 vm2@0-1.local
28.9 condor 6/8 10:49 0+00:06:15 vm1@reloaded
28.10 condor 6/8 10:49 0+00:06:12 vm2@reloaded
28.11 condor 6/8 10:49 0+00:02:48 vm2@0-9.local
```

En el archivo de emisión de un trabajo se puede también especificar la un archivo para llevar un registro de lo que sucede con el trabajo.
Cuando el trabajo termina, los resultados se envían a la máquina que lo emitió. Por lo tanto, si hay muchos trabajos ejecutándose, se necesita una gran cantidad de memoria real y de espacio de intercambio en el administrador central [5].

4.4 Checkpointing

El mecanismo de checkpointing permite obtener el estado actual de un programa a fin de que pueda ser reiniciado posteriormente desde ese último estado.
Condor a través del planificador puede quitar un recurso a un trabajo, efectuar el checkpointing del trabajo y reiniciarlo más tarde sin perder lo hecho hasta ese punto. Por defecto, se genera un archivo de checkpoint que se almacena en el disco local de la máquina que emitió el trabajo.

5 ENTORNOS DE EJECUCIÓN EN CONDOR

Condor ofrece varios entornos de ejecución (llamados universos) de los cuales podemos elegir a la hora de ejecutar un trabajo, y este universo elegido se especifica en el archivo de emisión del trabajo (ver Figura 2). Los distintos universos disponibles en Condor se comentan seguidamente:

Standard: permite checkpointing y ejecución remota de llamadas al sistema. Estas características hacen que un trabajo sea más confiable y le permite acceder de manera uniforme a todos los recursos del pool. Para que un trabajo se pueda procesar en el universo standard debe enlazarse con el comando `condor_compile`. Condor efectúa checkpointing de los trabajos a intervalos regulares. Si un trabajo debe migrarse de una máquina a otra, a través del mecanismo de checkpointing, Condor genera una imagen del estado del trabajo, copia la imagen en la otra máquina y reinicia el trabajo desde el punto en que se dejó de ejecutar. Los trabajos pueden ejecutarse en diferentes máquinas a lo largo de su vida, gracias a la ejecución de remota de llamadas al sistema.

Vanilla: este universo permite ejecutar trabajos que no pueden ser relinkeados contra Condor. En este universo es posible realizar checkpointing y ejecución remota de llamadas al sistema. Si un trabajo es terminado parcialmente, Condor puede suspenderlo para completarlo posteriormente o reiniciarlo en otra máquina del pool. Como no puede efectuarse la ejecución remota de llamadas al sistema, Condor trabaja con un sistema de archivos compartidos, como NFS o AFS. Alternativamente, Condor puede transferir los archivos que necesita el trabajo para ejecutarse.

PVM: permite la ejecución de programas escritos para Parallel Virtual Machine

MPI: permite la ejecución de programas escritos para MPICH.

Grid: brinda a los usuarios una interfaz Condor estándar para que puedan ejecutar trabajos en sistemas de administración remotos.

Java: un programa especificado para el universo Java puede ejecutarse en cualquier máquina con JVM, más allá de su ubicación, propietario o versión de JVM.

Scheduler: permite a los usuarios la ejecución de trabajos livianos en la misma máquina que los emitió. Este universo tiene posibilidades muy limitadas, por ello cuando el trabajo debe ejecutarse en la máquina que lo envía, se prefiere el universo local.

Paralelo: permite la ejecución de programas paralelos, por ejemplo los trabajos MPI.

Local: en este universo, el trabajo no espera asociarse con ningún recurso, ya que se ejecuta directamente en la máquina que lo emitió.

Cuando se prepara el archivo para emitir el trabajo a ejecutar se debe brindar información acerca del mismo, como cuál es el archivo ejecutable, los archivos que se necesitan como entrada, la plataforma necesaria para ejecutar el trabajo, la cantidad de veces que debe ejecutarse, etc.
Los agentes y los recursos anuncian sus características y requerimientos en anuncios clasificados (ClassAds). Condor, realiza asociaciones dando especial importancia a dos atributos especiales del archivo de emisión del trabajo: `Requirements` y `Rank`. `Requirements` indica requerimientos específicos y `Rank` indica preferencias, para que dentro de los recursos que cumplan los requerimientos, el trabajo se ejecute en máquinas determinadas.
El algoritmo de asociación necesita que ambos ClassAds coincidan y que los `Requirements` correspondientes sean evaluados como verdaderos.

6 RECONOCIMIENTO DE PATRONES EN EL TRÁFICO DE RED

6.1 Introducción al problema

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 [6,7,8].
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 [9].

6.2 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.

$$f(r) = \frac{\prod_{j=1}^{m} \prod_{i=1}^{n} \alpha(r_i, d_{ji})}{|D|} \quad (1)$$

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

$$\alpha(r_j, d_{ji}) = \begin{array}{c} w_i \text{ si } r_j = d_{ji} \\ w'_i \text{ si } r_j \neq d_{ji} \end{array} \quad (2)$$

Donde cada gen tiene un peso asociado w_i 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 w'_{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 [10]. El primer conjunto contiene 9000 entradas que representan 4 horas de tráfico, utilizado para la etapa de entrenamiento. El segundo conjunto contiene 35000 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. Si embargo al tratarse de una tarea inherentemente paralela, se puede pensar en la utilización de computación distribuida.

6.3 Implementación computacional

EL algoritmo fue implementado en Python [11] por tratarse de un lenguaje de alto nivel que permite desarrollar prototipos funcionales en poco tiempo. Al existir versiones de la maquina virtual de Python para múltiples sistemas operativos, esto permite la portabilidad.
La aplicación esta 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 maquina virtual los tiempos de ejecución resultan mucho mayores a los de un programa que se ejecuta de manera nativa. Para solucionar este problema se utilizo un compilador en tiempo de ejecución [12] (pysco)

6.4 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 interprete 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 [13]. Este archivo a su vez es enlazado contra una biblioteca estática que contiene la maquina 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 [14]. 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.

6.5 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 la misma maquina serial es cercano a las 7 horas.
En el trabajo serial [3] 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 3. 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 3 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

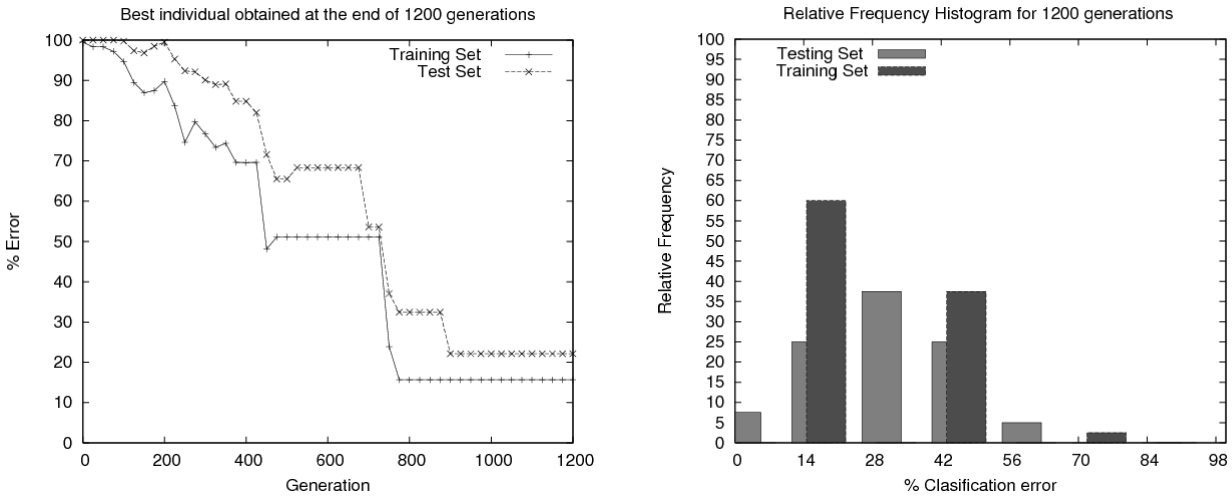

Figura 3. Resultados del mejor individuo obtenido y histograma de frecuencias relativas de la aplicación serial.

La posibilidad de ejecutar la aplicación del algoritmo genético en un entorno Condor ha permitido extender el número de generaciones del algoritmo a 1800. En este caso se utilizan todos los recursos computaciones disponibles de la Figura 1. 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 maquinas 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 1800 es de 20 minutos.

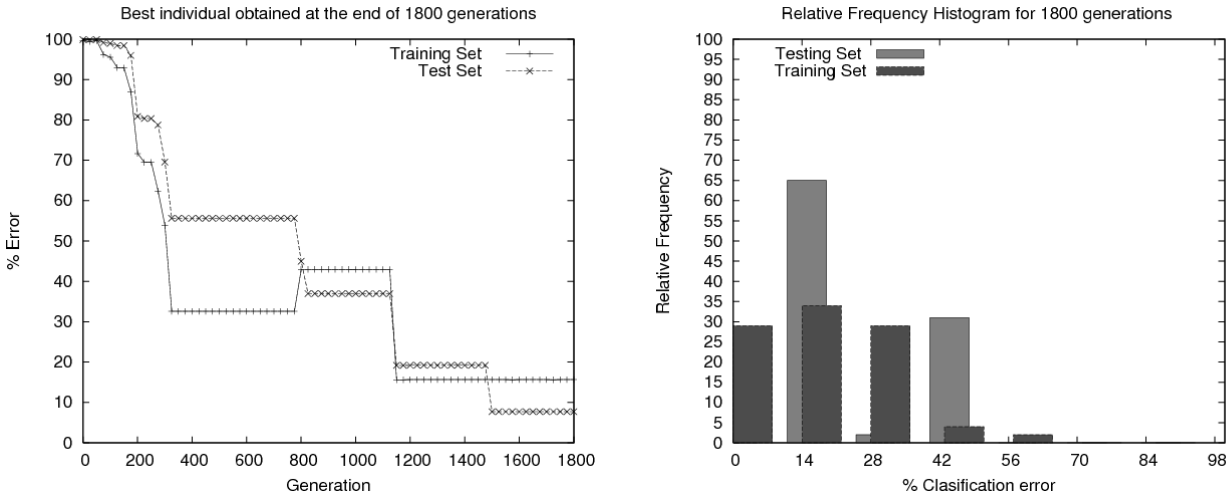

Figura 4. Resultados del mejor individuo obtenido y histograma de frecuencias relativas de la aplicación ejecutada en entorno Condor

Las pruebas preliminares del algoritmo en un entorno Condor han permitido mejorar los resultados anteriores. Luego de 1800 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%

7 CONCLUSIONES

En este documento se ha discutido la ejecución de una aplicación para la búsqueda de patrones de tráfico de red en un entorno Condor, con el objetivo de ser utilizados en Sistemas de Detección de Intrusos. La ejecución de esta aplicación ha permitido comprobar las capacidades de Condor a la hora de planificar y monitorear los trabajos y los recursos disponibles.

Condor brinda al usuario la posibilidad de ejecutar trabajos con requerimientos elevados, inclusive si los recursos no están temporalmente disponibles, ofreciendo capacidades únicas como migración de procesos y checkpointing.

Estas facilidades otorgan un alto poder computacional haciendo posible que cálculos con alto costo de gestión puedan ser realizados en forma ágil y eficiente.

Con una infraestructura de 12 máquinas bajo un entorno Condor, se obtiene una reducción del tiempo total de ejecución requerido por la aplicación. Por otra parte 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. Cabe destacar que la aplicación serie original se mantuvo sin cambios.

En un futuro, se planea configurar varios pool y administrarlos con Condor, así como también administrar máquinas con doble microprocesador, que permite dos instancias de trabajo a la vez.

REFERENCIAS

[1] Sanz A. Condor. Manual de usuario para el cluster HERMES, Instituto de Investigación en ingeniería de Aragón, Universidad de Zaragoza. Disponible en: http://i3a.unizar.es/hermes/manual_hermes.pdf, 2006.

[2] Thain D., Tannenbaum T., and Livny M., Distributed Computing in Practice: The Condor Experience, Computer Sciences Department, University of Wisconsin-Madison. Disponible en: http://www.cs.wisc.edu/condor/publications.html, 2002.

[3] 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.

[4] Livny M., Basney J., Raman R. y Tannenbaum T. Mechanisms for High Throughput Computing, Department of Computing Sciences, University of Wisconsin-Madison. Disponible en: http://www.cs.wisc.edu/condor/publications.html, 1997.

[5] Condor Team, Condor® Version 6.8.1 Manual, University of Wisconsin–Madison, Disponible en: http://www.cs.wisc.edu/condor/manual/, 2006.

[6] Bridges S. and Vaughn R. Fuzzy data mining and genetic algorithms applied to intrusion deteccion. In National Information Systems Security Conference, 2000.

[7] 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.

[8] Li W. and Traore I. Detecting new forms of network intrusion using genetic programming. Computational Intelligence, Vol 20, 475-494, 2004.

[9] Mukherjee B., Heberline L. T. and Levitt K. Network instruction detection. IEEE Network, 1994.

[10] Mit lincoln laboratory DARPA data set. Disponible en http://www.ll.mit.edu/IST/ideval/data/dataindex.html.

[11] Python Programming Language. Disponible en http://www.python.org.

[12] Rigo R. Psyco, the python specializing compiler. Disponible en: http://psyco.sourceforge.net/psyco.ps.gz, 2001.

[13] C. C. Ltd. cx freeze, a set of utilities for freezing python scripts into executables. Disponible en: http://python.net/crew/atuining/cx_Freeze/.

[14] Peter S. makeself, make self-extractable archives on unix. Disponible en: http://www.megastep.org/makeself/.